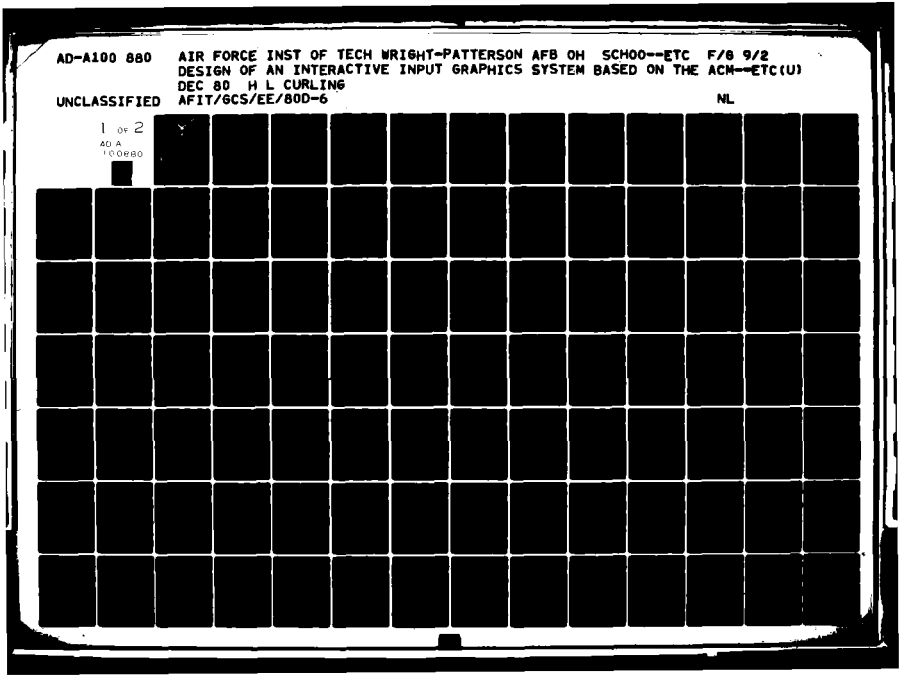
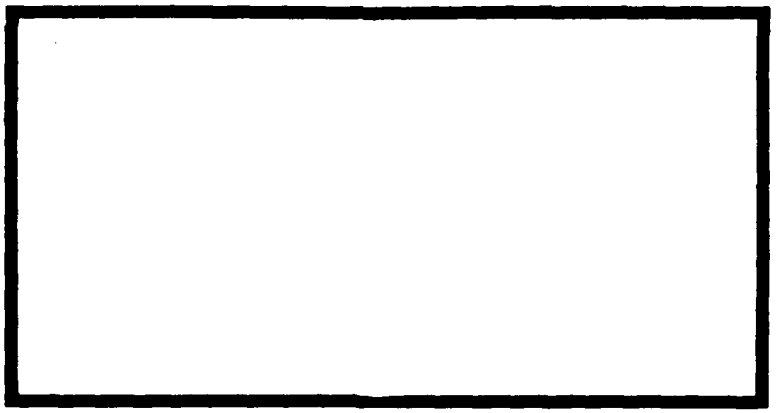

AD-A100 880
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 9/2
DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM--ETC(U)
DEC 80 H L CURLING
UNCLASSIFIED
AFIT/GCS/EE/80D-6
NL
1 OF 2
AD A
100880

LEVEL II

AD A100880

DTIC FILE COPY

DTIC
ELECTE
JUL 2 1981
S
A

UNITED STATES AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio



81 6 30 072

AFIT/GCS/EE/80D-6

# DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM CORE STANDARD

THESIS

AFIT/GCS/EE/80D-6 Harold L. Curling, Jr.
Capt USAF

AFIT/GCS/EE/80D-6

# DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM CORE STANDARD

THESIS

Presented to the Faculty of the School of Engineering

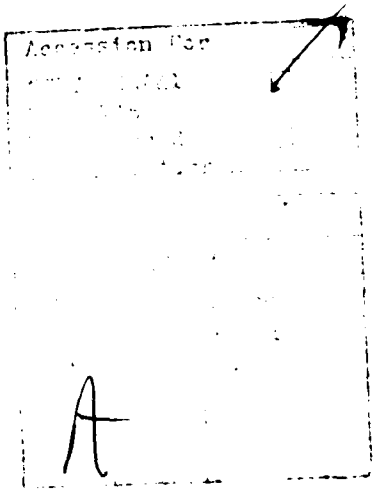
of the Air Force Institute of Technology

Air Training Command

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science




by

Harold L. Curling, Jr., B.S.E.E.

Capt USAF

Graduate Electrical Engineering

December 1980

Approved for public release; distribution unlimited.

## Preface

This report represents the design of a system of software routines required to implement the input half of the ACM/ SIGGRAPH CORE graphics standard. The design was developed with the goal of integrating this system with a system of graphics output routines designed independently by personnel of the Computational Services Division of the Air Force Weapons Laboratory. It is hoped that these designs will be integrated by follow-on efforts.

This thesis has been made possible with the assistance of several people whose help and support is gratefully acknowledged. Dr. Lamont has been an extremely understanding advisor. I would also like to thank the members of my thesis committe for being so understanding throughout this investigation. Most of all I wish to thank my wife, Becky, for all her support, encouragement and understanding during the past nine months. She has been my mainstay.

## Contents

## List of Figures

## Abstract

A design has been developed for the interactive input section of a graphics system based on the ACM/SIGGRAPH CORE standard of 1979. The input system has been designed for use on a Digital Equipment Corporation VAX 11/780 computer within the VAX/VMS operating system and contains the design for a device driver for the Tektronix 4014 graphics terminal. The design is readily expandable to include software drivers for other graphics devices. The Warnier-Orr diagram method of software design has been used as the vehicle for the development of the design. Three dimensional functions, the STROKE logical device and asynchronous event generation and data access are not supported by the design. However, all input functions required by the CORE standard have been included in the design.

# DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM CORE STANDARD

## I. Introduction

The purpose of this thesis investigation was to design an interactive system of routines whose purpose was to interpret and execute graphics commands generated from a user program. These commands would be generated by function calls within the user program and pertain only to input-oriented actions. These actions include, but are not limited to, acquiring data about graphics displays. This data might be the position of a cursor on the graphics surface or of a part of the display defined by the operator touching a light pen to the surface of the graphics screen.

The requirements for such a system of routines was first defined by the Association for Computing Machinery's standards committee in a report (CORE) published in the fall of 1977. This requirement was updated in August of 1979 by the same committee and contained major changes designed to increase the scope of the system. During the course of this thesis effort a design was developed based on the 1979 CORE report.

The following sections of this chapter discuss, in order, background information about the CORE and the need for an input graphics standard, the objectives of this thesis effort, the design approach used and an overview of the thesis.

## Background

The Computational Services Division of the Air Force Weapons Laboratory (AFWL) at Kirtland AFB, NM has been tasked with developing a computer graphics system which will be device-independent (both input and output) and host-independent. This system is to be used by applications programs for computer aided design. Therefore, it is required that the system be responsive to interactive environments.

There currently exists a device-independent output package for display of computer graphics. This package is called META and is now operating on the AFWL CDC CYBER 176 computer. The user program calls standardized subroutines in the META file system to display graphic information. Regardless of what device has been specified (in the installation job control language) the user program need not be modified. The META system interprets the user calls and generates a device-independent command string. This string of commands is then interpretted by the DIRECT interface which generates device-dependent commands to whatever display device (microfilm, graphics terminal, printer, plotter, etc.) has been specified in the JCL.

These interfaces are based roughly on the Association for Computing Machinery's SIGGRAPH CORE Standard. This standard has been developed in order to foster portability of user programs among different hosts and devices.

Development of a CORE-based input system package for the META system has not yet been attempted and is the main impetus for this investigation.

The CORE Standard was first published in 1977 and revised in 1979. Developed by the Special Interest Group on Computer Graphics (SIGGRAPH) of the Association for Computing Machinery (ACM), its purpose is to allow users to write graphics programs which will execute on different hosts and which will interact correctly with all graphics input and display devices equally well.

The intent of the Standard is to foster portability of user applications programs. Portability, as defined by the CORE Standard is "...the ability to transport graphics applications from one installation to another with minimal changes." As a further expansion of this portability definition it is intended that user programs execute equally well in combination with any input and/or output device.

As an example of device-independent output. The META file system and the DIRECT interface will cause the applications program graphic subroutine calls to be translated into correct commands for a line printer plot, a graphics terminal display or a hard copy plotter.

The following is an example of device-independent input. The user program may request input information from the operator. If the terminal is equipped with a light pen, the CORE Standard based system might respond by flashing a menu on screen and directing the operator to select an option by 'penning' it from the menu on screen. If this same request were encountered while the operator is communicating to the program through a terminal such as the TEKTRONIX 4014, the system might respond by displaying a menu on screen and request the operator to choose an option and type an appropriate response on the keyboard.

This total graphics system will execute interactively with an operator in order to facilitate applications such as computer aided design (CAD). This, however, will not restrict the system from operating equally well and efficiently in output-oriented applications such as hard copy plotting and microfilming.

## Objective of this Investigation

The objective, then, of this effort is to design an interactive graphics input system using as a specification, the ACM SIGGRAPH CORE Graphics Standard. Emphasis will be placed on a top-down structured design with the AFIT/ENG Digital Equipment Corp. VAX 11/780 as the target computer system.

## Approach

This investigation involves three major steps. These steps are the requirements definition, choice of a software design system, and the actual design itself.

The CORE standard served as the requirements document for this investigation. The standard is embodied in the 1979 CORE standards committee report.

The choice of a software design technique or system was made after surveying many techniques available and analyzing three of the techniques best suited, in the author's opinion, to accomplish the CORE system design.

## Overview of the Thesis

This investigation involved the analysis of the system requirements document (CORE standard) and the implementation of the requirements using the Warnier-Orr technique of top-down structured design. Appendix A contains a list of the CORE-defined routines. Appendix B contains the data structure diagrams and Appendix C contains the process structure diagrams for the design.

The thesis is arranged by chapters which follow the general approach used in the thesis investigation. This chapter contains the introduction to the problem and background information relating to the problem. Chapter II gives an overview of the requirements document for the investigation while Chapter III discusses the software design techniques investigated and a discussion of the choice of the technique used for this thesis effort. Chapter IV presents considerations which bore on the design effort while Chapter V details the actual system design. Chapter VI presents the conclusions and recommendations of this investigation.

## II. An Overview of the CORE Graphics Standard

### Definition and Motivation

In the fall of 1977, the Graphics Standards Committee of the ACM/SIGGRAPH introduced an inital report (CORE) on the standardization of input and output for computer graphics generation, display and alteration [Ref 2]. In August of 1979 a revised report was published [Ref 1]. This revised report incorporated new capabilities and routines to support computer systems which had been excluded from support by the original CORE system. This exclusion occured because of operating system restrictions or lack of capabilities. These restrictions occur in the area of interrupt-driven routines or asynchronous operation of the CORE system (e.g. event handling routines). To preserve the characteristic of flexibility the original CORE standard required operator-generated events to be queued independently of the user program. This provided for multiple, simultaneous (or near-simultaneous ) event generation such as might occur in an aircraft simulator (stick, rudder, peddles, etc.). Some operating systems do not have this capability and the revised standard provides for synchronous operation of the system.

The CORE standard strives to create a device-independent environment for the generation, display and alteration of computer graphics. This is extended to host computer independence by the use of standard high order languages (HOL) such as FORTRAN and PASCAL for the coding of the user's application program. User program portability is the main goal of the CORE system.

The CORE system of routines is logically divided into two sections: input and output primitives. Primitives, used in this sense, refers to the highest level routines which accomplish the specific functions and tasks of the standard; that is, those routines directly called by the applications program.

The output section of the CORE system is composed of those routines whose function is to display graphic information on the viewing surface independent of the relationship between the view surface logical device and its physical device. A number of attempts have been made to accomplish this implementation. The most notable, and the one that most closely adheres to the specifications as well as the spirit of the CORE standard, is the Lawrence Livermore Laboratory (LLL) GRAFLIB library of routines [Ref 7 and 8].

The GRAFLIB package diverges from the output section of the CORE standard in function naming conventions only. The output routines were given different names according to the LLL group's own conventions but in all cases there is a clear cross-reference to the CORE system explicitly in the GRAFLIB user's manual. All formal parameters of each routine agree in position, function and number, if not exactly in name.

Another attempt was made by the computational services division of the Air Force Weapons Laboratory (AFWL/ADD) at Kirtland AFB, New Mexico. However, this system predated the CORE standard by a number of years and, therefore, is less faithful to the specifications of the CORE. This system, called META, uses an intermediate command file, created by calls from the application program. This command file interfaces to the device-dependent routines which display the graphic data on the physical device(s). The data, in this case, is defined by the command file.

The input section of the CORE, unlike its output counterpart, has not yet been implemented or designed for any system. Implementation of the output section is necessary in both batch and iteractive computing environments. Implementation of the input section is necessary only in the interactive environment. Since the batch environment is predominate in systems which are large enough to make a standard graphics package desirable, attention has been channeled into output rather than input. One of the main reasons for this project is the growing interest within the Air Force in interactive graphics (e.g. simulator displays).

The input section facilitates the generation (definition) and alteration of graphic displays. The input system has associated with it logical devices and input primitives. The logical devices are discussed next followed by their relationship to the user program through the input primitives. The input section of the CORE provides the interface between the operator and the display through the use of logical input devices.

CORE Input System Logical Devices

The input system logical devices are divided into two categories: event devices and sampled devices. This divides the logical devices into two groups composed of functionally similar devices, permitting one routine (e.g. AWAIT_EVENT) to handle data acquisition from logically different devices. The event devices are those that signal events to the user application program and are generated by the operator. The sampled devices are those which the application program polls for information. Event devices can not be sampled and sampled devices do not cause events.

Event devices cause the generation of "event reports". These reports are placed on the "event queue" and are then available for interogation by the application program. These event reports contain information about which specific device caused the event and such information as may be necessary for the application program to process the event.

The classes of logical input devices and their functions are as follows:

| Event Devices | Sampled Devices |
|---|---|
| PICK | LOCATOR |
| KEYBOARD | VALUATOR |
| BUTTON | |
| STROKE | |

A logical device class must be implemented either physically or by simulation. A physical implementation is achieved when a logical device has a direct physical counterpart (e.g., the terminal KEYBOARD). Simulation of a logical device must be implemented when this physical equivalence is not possible. In the case of the Tektronix 4014 terminal used for this effort, a PICK logical device has no physical counterpart. Therefore, it is simulated with the 4014 cross-hair and LOCATOR thumbwheels.

These logical input devices are defined by their purpose, their typical physical device, their simulator device(s), what the device initiates (event devices only) and what echo on the view surface, if any, will be used. An echo is the visual representation provided for

the operator by the CORE system, of actions being taken by the operator and the CORE system response to those actions.

The following discussion delineates each logical device class with respect to these defining attributes.

PICK - this is a logical device which returns (in an event report on the event queue) the identification of the segment and primitive "picked" by the operator.

NOTE

A segment is a portion of the output display which defines a unique set of primitives and a primitive, in this sense, is a drawn line or a string of text.

Typical device : light pen.

Simulator device : data tablet, joystick and a hardware or software coordinate comparator.

Causes : a logical event report to be added to the event queue.

Echo : blink the segment "picked"

KEYBOARD - a device which provides a means for the operator to enter characters or text strings.

Typical device : alphanumeric keyboard.

Simulator device : light buttons and a light pen or a data tablet.

Causes : an event report to be added to the event queue ; the report contains the character/text string typed and the number of

characters in the string.

Echo : display character/string typed at a predetermined position (determined by application program) as operator types them.

BUTTON - a logical device which provides the operator with a method of specifying applications program functions.

Typical device : physical button on program function keyboard (pfk)

Simulator device : light buttons or by keyboard.

Causes : a logical event report to be added to the event queue. The event report contains the ID of the button pressed.

Echo : lamps under the physical buttons or display on terminal.

STROKE - device that allows the operator to draw a line on the view surface by specifying a series of positions (points) on that surface.

Typical device : a data tablet stylus.

Simulator device : there is no convenient simulator for the STROKE device.

Echo : display the line drawn on the view surface and highlight the points in the series.

LOCATOR - a logical device which provides coordinate information in normalized device coordinate space (that space defined by the user application program). This information represents a position on the view surface.

Typical device : data tablets and joysticks.

Simulator device : light pen tracking cross.

Echo : a cursor (typically a cross) displayed at the current LOCATOR position.

VALUATOR - a sampled device which provides the operator with a means of entering numeric scalar values to the applications program within a range of values specified by the user.

Typical device : control dial (analog) ; however, multi-position switches and toggle switches may be used (digital).

Simulator device : light pens, data tablets or keyboard (typing the value desired) in conjunction with displays on the view surface.

NOTE

VALUATOR values are constrained to lie within a program-specified range. The CORE system maps the physical device values linearly into this range. The initial value is specified by the user application program.

Echo : the numeric values are displayed on the view surface at an application program-specified position (updated constantly to reflect the VALUATOR's current value.

## The Minimum Set of Logical Devices

The CORE system requires that any implementation of the standard incorporate a minimum set of logical input devices in order to preserve a "reasonable level of portability" [Ref 1 and 2]. Portability is defined as the ease with which a user program can be moved from one computer system to another. without modification. A highly portable system requires few or no changes to the user program in order to run on different implementations of the CORE system. Obviously, the fewer changes required of the user the more portable the CORE implementation is. The CORE standard is the result of the desire for portability. This minimum set is composed of the following devices either explicitly or by simulation:

1 PICK device

1 KEYBOARD device

1 STROKE device

1 LOCATOR device

4 VALUATOR devices

8 BUTTON devices

The CORE system further requires that the four VALUATOR devices support a minimum of six bit precision. This is an arbitrary requirement inflicted by the CORE standards committee.

## Input System Primitives

The input primitives of the CORE system are those application program- callable routines which accomplish the graphics input functions.

The CORE system supports a set of primitives grouped into nine categories:

1. Initializing and Enabling devices

2. Reading SAMPLED devices

3. Event handling

4. Associating devices

5. Accessing event report data

6. Synchronous input functions

7. Device echoing

8. Setting input device characteristics

9. Inquiry

The specific routine names, their formal parameter lists and the error returns generated are listed in Appendix D. The following discussion details each category.

Initializing and Enabling Devices - The two functions of initialization and enabling are very much alike. Initialization insures that the applications program will have access to the device and enabling allows input from the device. Once a device has been initialized and enabled event reports are allowed for event devices and

sampling is allowed for sampled devices. When a device has been disabled or has not been initialized event generation/sampling is not allowed. However, once event reports are placed on the event queue, disabling of a device produces no effect on the event queue.

Reading SAMPLED Devices - The routines in this category provide for the reading of information from LOCATOR and VALUATOR devices providing, of course, that those devices have been enabled.

Event Handling - The routines which make this category provide the operator with a means of causing events within the applications program. This is accomplished by the CORE system by placing an event report on the event queue for the user program to poll. Event handling is the most difficult part of the CORE system for most computer operating systems to incorporate. This is due to the fact that most operating systems will not support interrupt-driven routines. For this reason, the Standards Committe of the ACM/SIGGRAPH included synchronous input in the 1979 revised CORE standard.

Associating Devices - Many times, if not every time, events occur more information will be required than is available from the event report of the device alone. However, no event device may be sampled. Therefore, it is possible to associate sampled devices with event devices. When an event occurs, the associated sampled devices are sampled and that information is included in the event report. Only sampled devices which are enabled will contribute any information.

Accessing Event Report Data - When the event handling routines de-queue the top report from the event queue, the information in that report is copied into the current event report. This report is then read by the applications program by the routines contained in this category.

Synchronous Input - This category has been added to the CORE system to facilitate the implementation of the standard within operating system environments which do not allow user level interrupt-driven routines. It also provides a method of efficiently handling situations where the physical devices employed allow only one event to take place at a time. This is the case with devices such as the Tektronix 4014 and 4014-1 graphics terminals.

This greatly simplifies an otherwise complicated implementation on systems where the full capabilities of the CORE system are not required o. are not supportable.

Device Echoing - The routines in this category allow the operator (user applications program) a limited control over the types of echo that the input devices exhibit. These echo options are included in Appendix D, also.

Setting Input Device Characteristics - The logical input devices have certain characteristics which describe how they will perform and within what ranges of values they will operate. The routines of this category give the applications program active control over these characteristics. If the specified device is enabled the setting of characteristics occurs immediately. If the device is disabled the characteristics take affect as soon as the device is enabled.

Inquiry - The routines contained in this category simply allow the applications program to access the current capabilities and characteristics of the logical input devices.

Observations and Comments

When this thesis project was initiated it was hoped that the VAX 11/780 computer system and its VMS (Virtual Memory operating System) would be capable of supporting the full CORE system and that a number of different physical devices would be available for interface. It has become apparent that neither of these conditions will be met. The operating system is capable of supporting a limited asynchronous version of the CORE and the only physical device available for interface is the Tektronix 4014 graphics terminal. For these reasons and those imposed due to time constraints it was decided to design for a synchronous implementation for the 4014 terminal only. This design will, however, support the full capabilities of a level two implementation (synchronous input).

NOTE

There are two other levels of CORE implementation: level one and level three. Level one allows no input (output only) and level three is full input (asynchronous).

## III. The Design System

The purpose of this chapter is to discuss the investigation of design techniques available for software developement, to analyze three different candidates for use as a design system and to discuss the choice of the technique used.

### General Comments on Software Design

Software design, or software engineering, is a young discipline. It has been only recently (since about 1972) that the need for software engineering has been recognized. However, there is no lack of software engineering methods available. Structured Analysis and Design Technique (SADT, a trade mark of the SofTech, Inc.), the Yourdon-Constantine Structured Design Method, the Michael Jackson method and Warnier-Orr diagrams are just a few of the methods available for designing software.

The number of methods available is large, yet they all share common goals:

1. Efficient implementation - efficient both in coding time and in number of statements coded.

2. Error reduction - logical and design errors.

3. Modularity of programs - this goal encompasses the aspects of cohesion (how well a module is tied together functionally) and coupling (how closely a module is related to other modules in the program).

4. Portability of design - if the actual statements of the program are not portable among different computer systems, the program design can be easily transformed into a usable program on any machine.

5. Program documentation - using a structured approach for designing computer software will produce a self-documented design. The design will become the documentation for the code.

Much time and effort must be exerted during the set-up phase of a structured design aproach. The software requirements are first translated into a structured description of what steps are necessary to accomplish the requirements. This is the basic first step no matter what design technique you choose. However, the rewards of investing the time and effort in a structured design more than compensate for this front-end loading. The most important of these rewards are ease of coding, error reduction (which aids in shortening the test phase), decreased effort needed to maintain the software once it has been produced (in terms of both debugging and updating), increased reliability and, of course, decreased software development costs.

Design Options

The set of methods chosen as being most applicable to this design effort includes: SADT, Jackson's method and Warnier-Orr diagrams. The list of options was limited to these methods due to time constraints (there simply was not enough time to investigate a large number of techniques) and due to fact that these methods are well known and have been widely published and utilized.

The discussion which follows describes each of the three techniques in terms of how they are used, their formats, their advantages and their disadvantages.

SADT

Structured Analysis and Design Technique is a trade mark of SofTech, Inc. [Ref 12]. The technique has been used successfully by many companies (including IBM, MITRE Corp., and Xerox Corp. to name just a few) and in Air Force programs to aid in the design of software systems. The top-down design technique employed in SADT is characterized by the use of control and data flow constructs combined to describe the interaction of functional modules within the program.

According to SofTech [Ref 12 :2-1] there are seven fundamental concepts of SADT. Briefly, these are:

1. SADT builds a model of the problem. Several models, from differing viewpoints, may be necessary to completely describe a problem.

2. Analysis of the problem is top-down, modular, heirarchic and structured.

3. SADT creates, first, a functional model of system performance and, second, a design model of how the system will be implemented to perform those functions.

4. SADT models both data and procedures and must show both models properly related.

5. The SADT language is a diagramming technique showing component parts, or modules, their interrelationships and how they fit into the heirarchical structure.

6. SADT supports coordinated teamwork.

7. SADT requires that all decisions (both analytic and design) and comments be made in written form.

Concept two, top-down structured decomposition, is the most important concept of SADT; indeed, of all the techniques which will be discussed.

The diagrams produced by this method are composed of two basic components: boxes and arrows. A set of boxes and arrows comprise an activity diagram which becomes one box in the next higher level of structural hierarchy. The boxes represent process modules and the arrows represent module interconnections.

There are four types of module interconnections: input, output, control and mechanism. The mechanism interconnection is seldomly used and nothing more will be said of it in this analysis. Figure 1 [Ref 12] shows the box and interconnection conventions and Figure 2 [Ref 12] shows an example of a typical activity diagram. A unique and self-explanatory name must accompany every box and arrow.

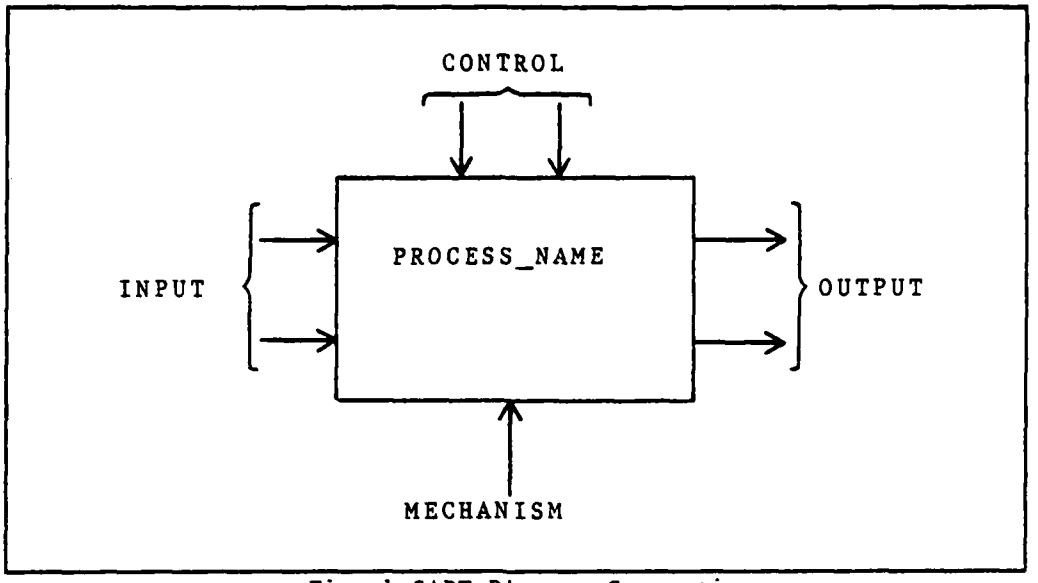



Fig. 1 SADT Diagram Conventions

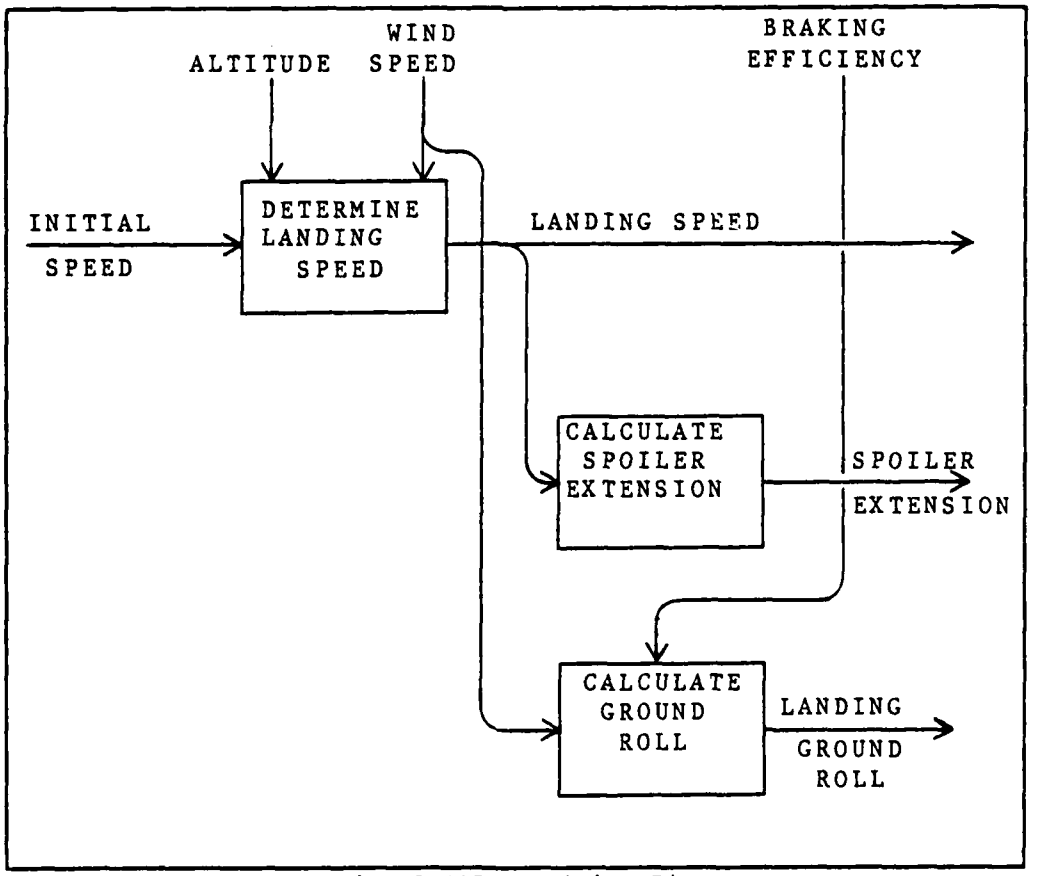



Fig. 2 SADT Activity Diagram

Any box may have multiple input, control and output arrows, as shown in Figure 1, or they may be "pipelined". Pipelining is depicted in Figure 3. The "pipe" consists of two or more related interconnections in any combination. This means that data and control arrows may be combined in the same pipeline. At any point on a diagram a part or parts of a pipeline may break away from the main pipe or may join it. When part of a pipeline splits from the main pipe that part must be identified by its name written on the arrow. A cross-reference of every pipeline showing its constituents and span of application (which activity diagrams it appears in) aids in understanding the diagrams and program flow.

The prime advantage of SADT is its highly structured, top-down format. This format provides understandable decomposition of the program functions.

SADT's disadvantages are few but significant. The diagrams are difficult to read. The ambiguity between data and control arrows causes confusion. Data and control flow become intermeshed. The pipeline concept, when used, causes the true nature of the arrows to be hidden. This is especially confusing when arrows split from or join pipelines from module to module and level to level. The SADT technique itself is structured but the diagramming conventions are not.

Jackson's Method

Michael A. Jackson has developed a method of structured program design called, simply, Jackson's method [Ref 9]. The separate components of Jackson's method are data structures, program structures and a list of the executable operations required.

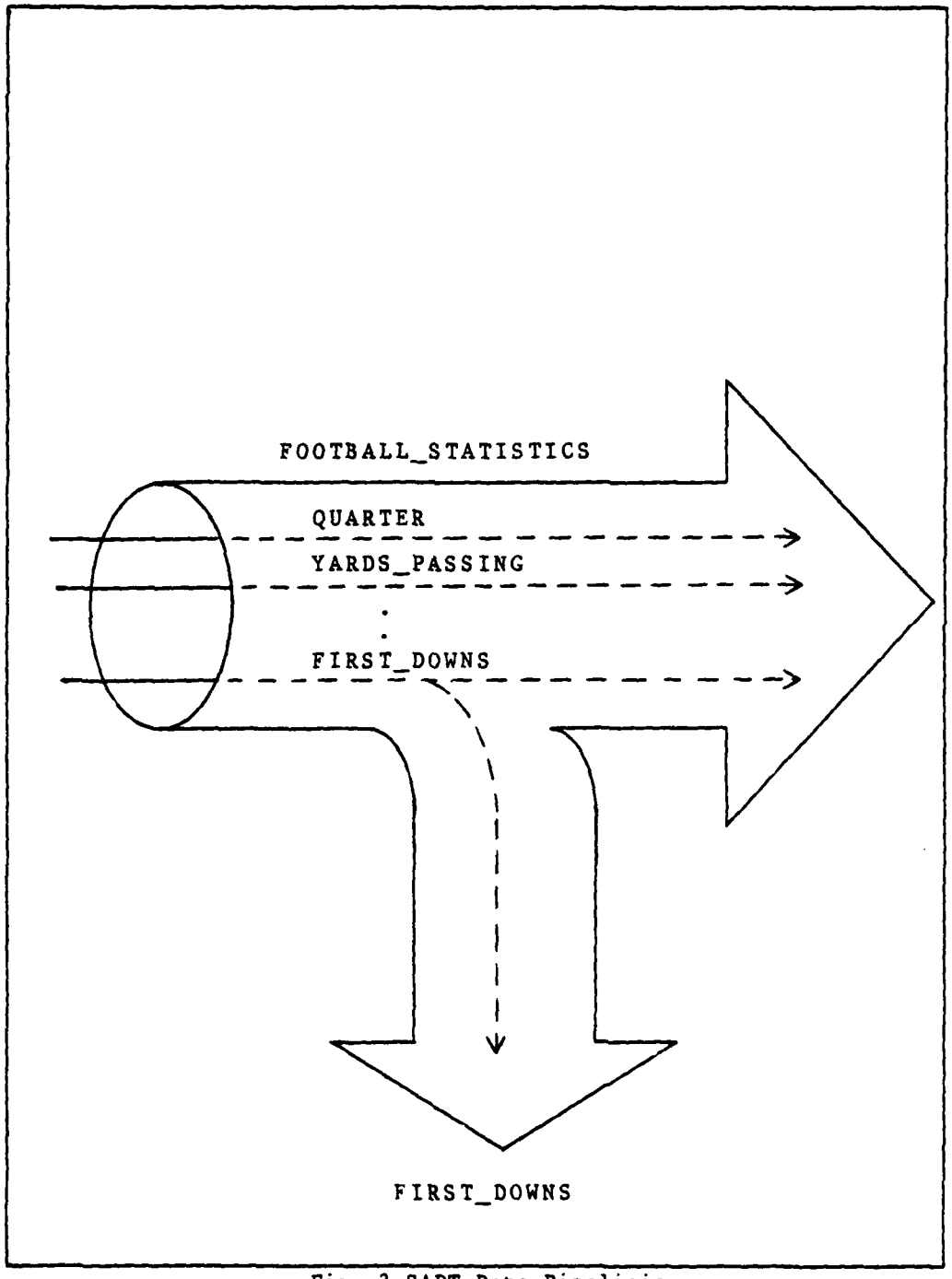



Fig. 3 SADT Data Pipelining

Jackson divides the problem into the two distinct structures of data and program. The structures are composed of modules; each module relating directly to a program subroutine counterpart in the final code. Both data and program structures are, like SADT, embraced by a top-down hierarchy.

The data modules can be files, records or individual elements ; with files in the top levels of structure and elements in the lower levels. There are normally two separate data structures : an input structure and an output structure. Depending on the problem, there will be one or more of both types of structures.

The program structure modules are one-to-one related to the data structure modules. In fact, the first step in the Jackson method as describe in detail later in this discussion is to produce a data structure and then fit the program structure around it. The program structure incorporates both the input and output data structures. Therefore, the program structure relates input to output.

Jackson's method follows a three step procedure. The first step is to generate all of the data structures in a top-down block diagram such as shown in Figures 4 and 5. The blocks of the diagram have special notations in the upper right corner. An asterisk (*) denotes that the block occurs repetitively. An "0" denotes that one and only one of the siblings of one parent at a particular level is included at that level at any one time. Absence of any notation signifies that the block is always included at that level once and only once.

The second step in the process is to produce a program structure that matches the data structure. In other words, the program structure is built around the data structure. An example of a program structure for the data structure of Figure 5 is shown in Figure 6.

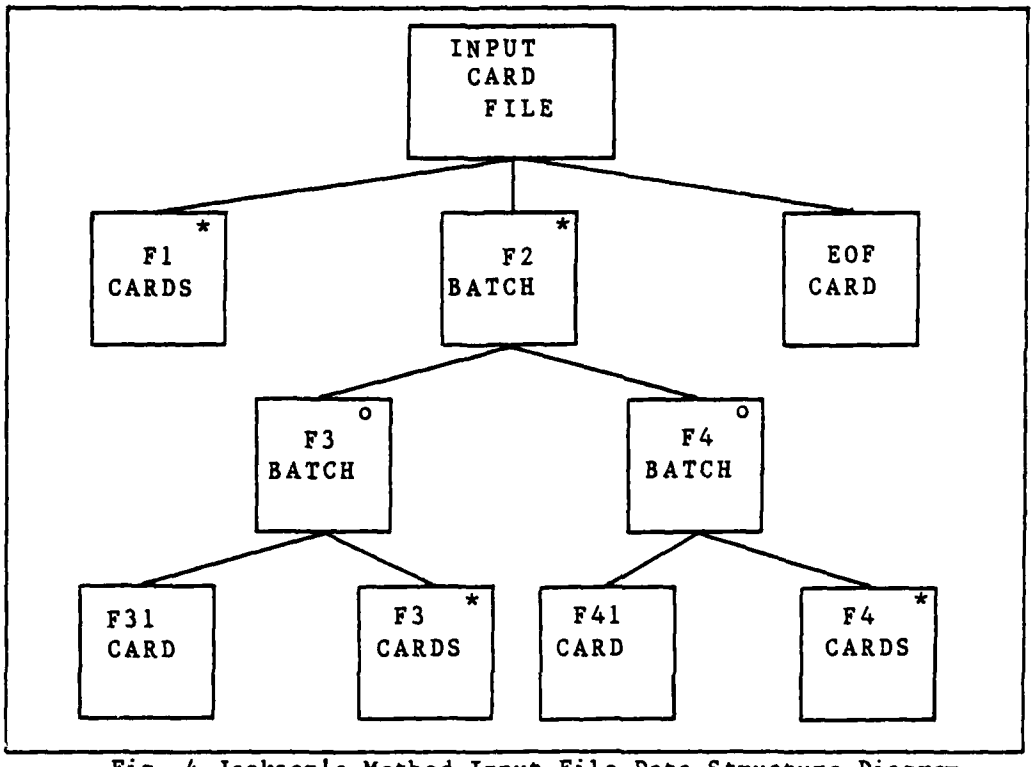



Fig. 4 Jackson's Method Input File Data Structure Diagram

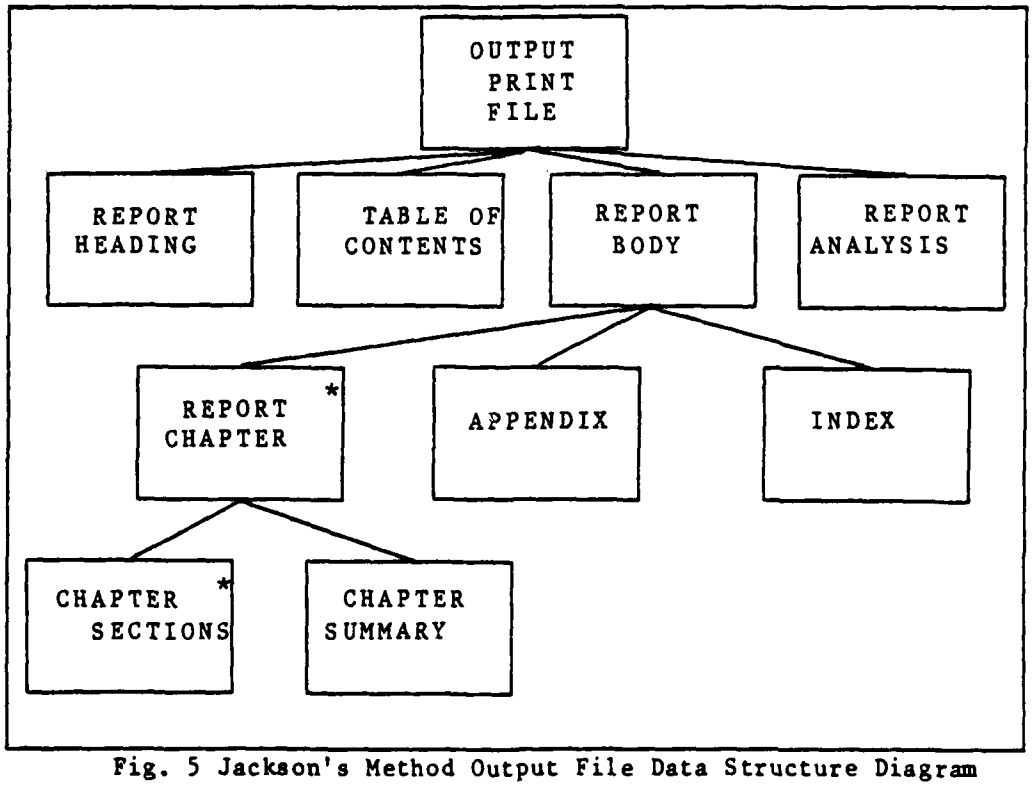



Fig. 5 Jackson's Method Output File Data Structure Diagram

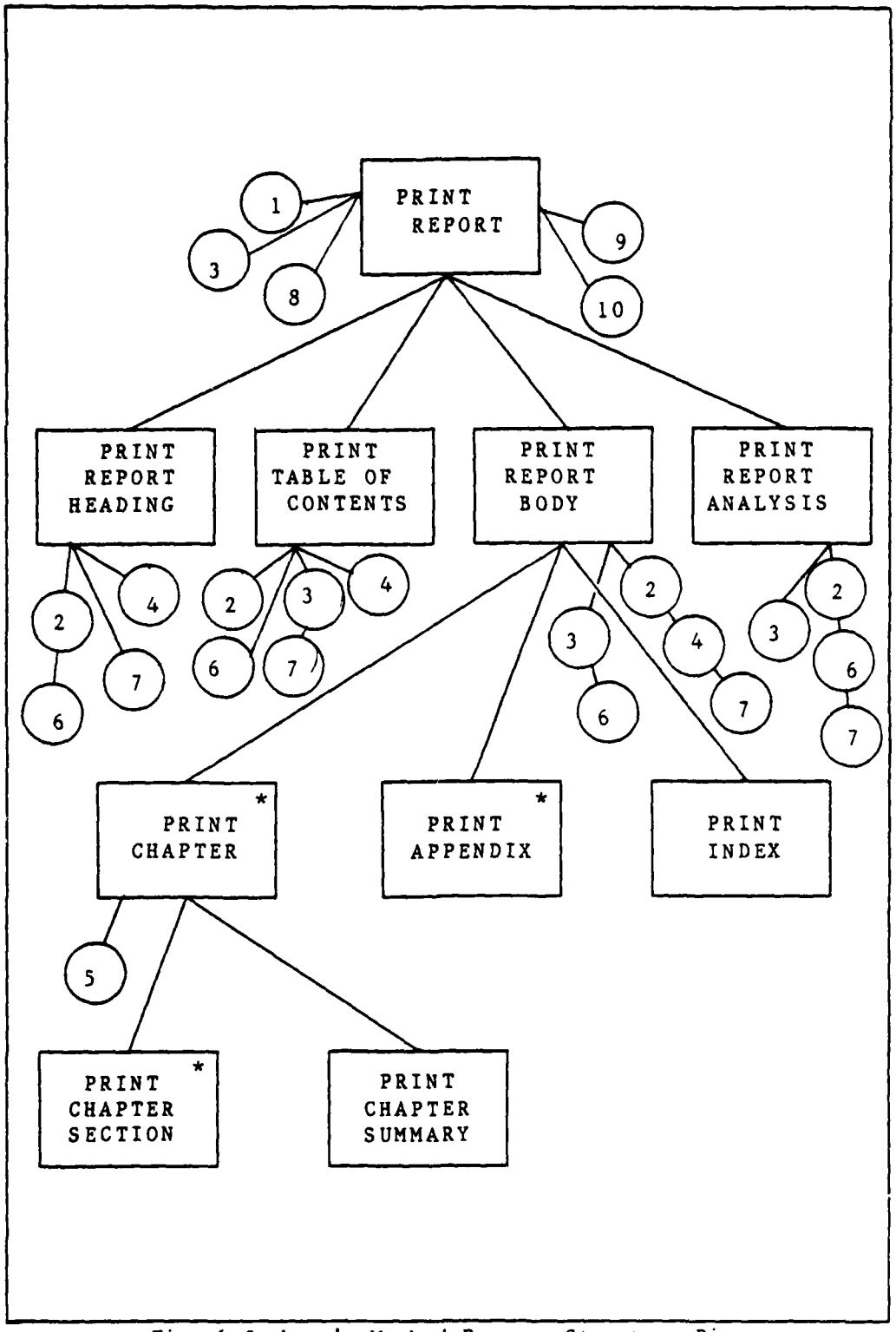



Fig. 6 Jackson's Method Process Structure Diagram

The same conventions of notation apply to the program structure as apply to the data structure : the asterisk implies that a module is executed repetitively, an "0" implies that one and only one of the siblings marked with this notation is to be executed and an absence of notation implies that the module is to be executed but only once.

The final step in Jackson's method is to generate a list of all executable operations required by the program structure and to allocate each operation to its proper place in the structure. The following is a list of required operations for the example process structure of Figure 6. The operations are shown properly related to the processes in Figure 6.

1. Initialize printer.
2. Move print line to printer.
3. Move page eject to printer.
4. Page <-- Page + 1.
5. Chapter <-- Chapter + 1.
6. Move character to print line.
7. Move line feed to printer.
8. Page <-- 0.
9. Line <-- 0.
10. Chapter <-- 1.

There are three important advantages that Jackson's method offers over SADT. The principal advantage is that Jackson's method provides unambiguous diagram conventions. Another advantage to the Jackson method is that the data, as well as the program, is structured. Therefore, both process decomposition and data decomposition are simpler and easier to understand. The third important advantage is that data and process module is annotated to show its usage in the context of the

structure. This adds clarity to the overall function of the program.

The main disadvantage of Jackson's method is the requirement that the process structure map one-for-one into the data structure. This requirement is often difficult to comply with and leads to unnecessary complications in the design. It is not always possible to make the program structure mirror the data structure, nor is it always desirable.

Warnier-Orr Diagrams

The final method considered as a possible design methodology is the Warnier-Orr diagram technique (after Jean-Dominique Warnier and Keneth T. Orr). The Warnier-Orr technique is a method of developing logically correct programs from the basic data structure of the problem [Ref 10 : 72]. Unlike Jackson's method, the Warnier-Orr approach provides a way of building program structure to fit the basic data structure of the problem without requiring that the program structure be a mirror image of that data.

The Warnier-Orr method is described in a four step procedure [Ref 11]. This step-by-step approach will lead to logically correct programs and an efficient design.

Identify the Output - The first step in any design should be to identify what the program is going to produce as its output. Every program must produce output in some form whether it is printed lines, control signals or graphic displays. A program that produces no output is a useless program ; it does nothing.

Another reason for defining the program output first is so that the end of the program function can be determined. In this way unnecessary programming effort is avoided. At the same time this insures that all of the requirements of the problem have been met.

A programmer makes his or her job unnecessarily hard if he or she attempts to design a program without knowing what output is needed or desired.

Define the Logical Data Structure - The second step of the Warnier-Orr method develops the logical grouping of data according to the way the data is related. The data is related in a hierarchical manner. Hierarchy in the Warnier-Orr diagram is defined as left (highest level) to right (lowest level), each level being separated by brackets. An example of a simple logical data structure is depicted in Figure 7 [Ref 11:61].

It must be emphasized that this logical data structure is not the detailed data structure, but only a graphical representation of how the data is logically related. There are many details in the data structure which are missing at this point in the design process but are supplied in the next step.

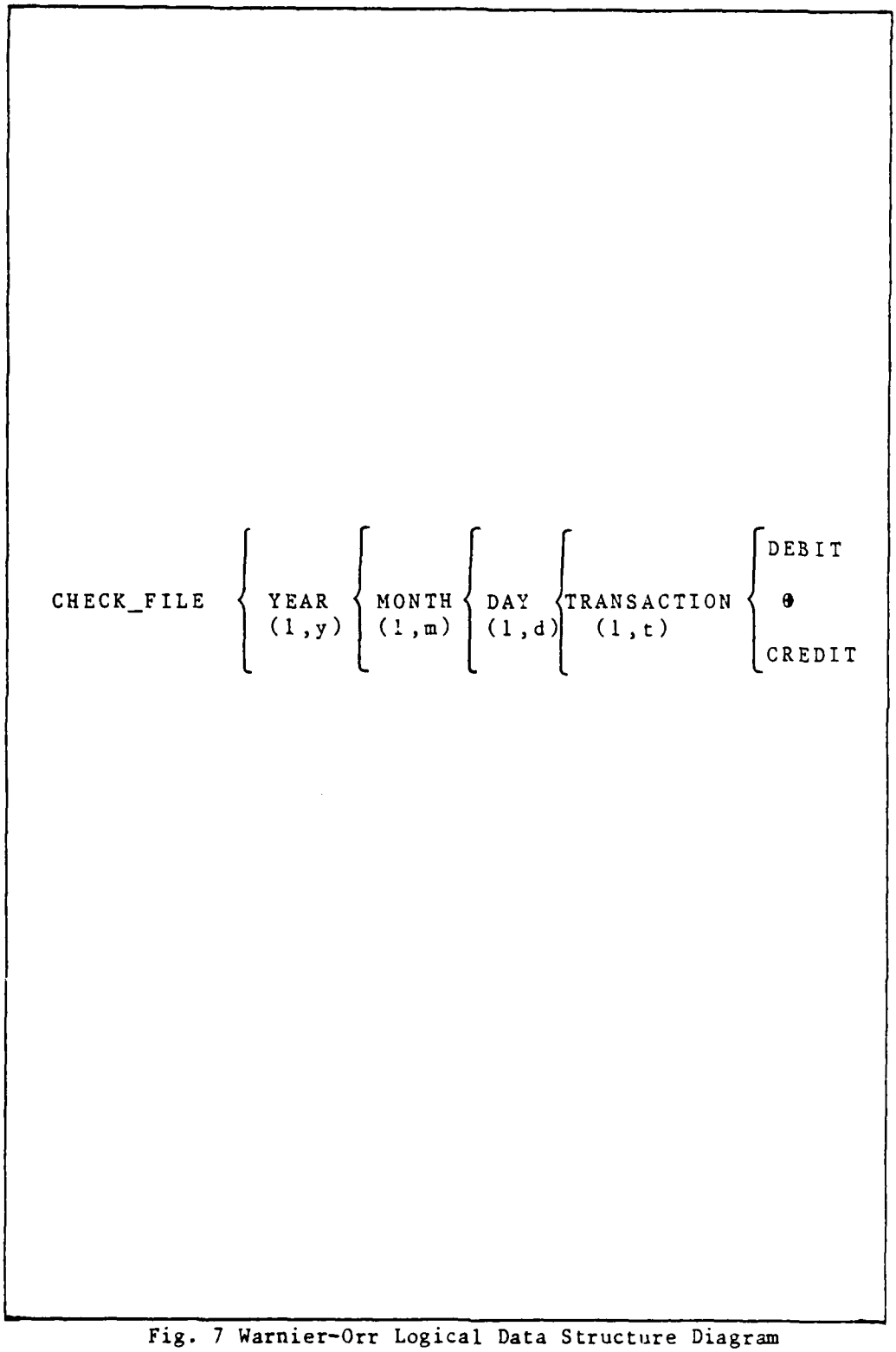



Fig. 7 Warnier-Orr Logical Data Structure Diagram

Define the Physical Data Structure - The form and detail of the physical data structure is inherently tied to the physical device or devices on which the structure is impressed. In other words, the organization of the pysical data structure is tailored to the device on which it must be stored.

Normally, the physical data structure mirrors the logical data structure. In this way, no transformation of data in form need take place between program usage and data storage. This will always be the case when the structure is stored entirely within the program structure. This implies that the data is stored totally in memory (e.g., does not need to be swapped between memory and external devices such as disks and magnetic tapes). This mirroring fits the direct access characteristic of memory.

If, however, the data necessarily will be swapped between memory and an external device (especially sequential devices such as magnetic tapes and certain disks), the physical data structure will not exactly mirror its logical counterpart. The difference between the logical and physical data structures will, in this case, be dictated by both the program usage and physical storage medium for the data.

Effort and time is needlessly wasted in the design phase and eventually in program implementation if the physical aspects of the data base are not accounted for in the design. Emphasis on this step will insure that a logical "blueprint" for the process structure is provided for the final step.

Design the Process Structure - The final step in the Warnier-Orr method is to define the program or process structure. If the other three steps in the technique are executed as outlined, this step of the procedure is greatly simplified. The detail attained in the first three

steps will dictate the degree of simplicity in this final step. During this step the process structure will be developed based on the physical data structure.

The process structure is developed hierarchically from left to right with brackets (as in the data structures) denoting separate levels in the hierarchy. Within each level (bracket) the program control flows from top to bottom and then begins at the top again if the process repeats.

An example of program structure is depicted in Figure 8. This process will insure that cohesion will be strong and functionally directed and that coupling between modules will be minimized. Strong cohesion and minimal coupling are attributes of a well designed program and will result in efficient program implementation. This is what is meant by a logically correct design.

As with all design methodologies discussed in this report, the Warnier-Orr method requires that names of processes and data elements indicate their function. Functionally descriptive names eliminate confusion and heighten understanding of the design.

The primary advantages which the Warnier-Orr technique posesses, with respect to SADT and Jackson's method, are simplicity, readability, and ease of implementation. The design proceedes with one step leading directly and logically to the next.

The design process stops when the last program hierarchical level defines one specific function or when the last level of data structure defines a unique element which cannot be further divided. These criteria are readily identifiable.

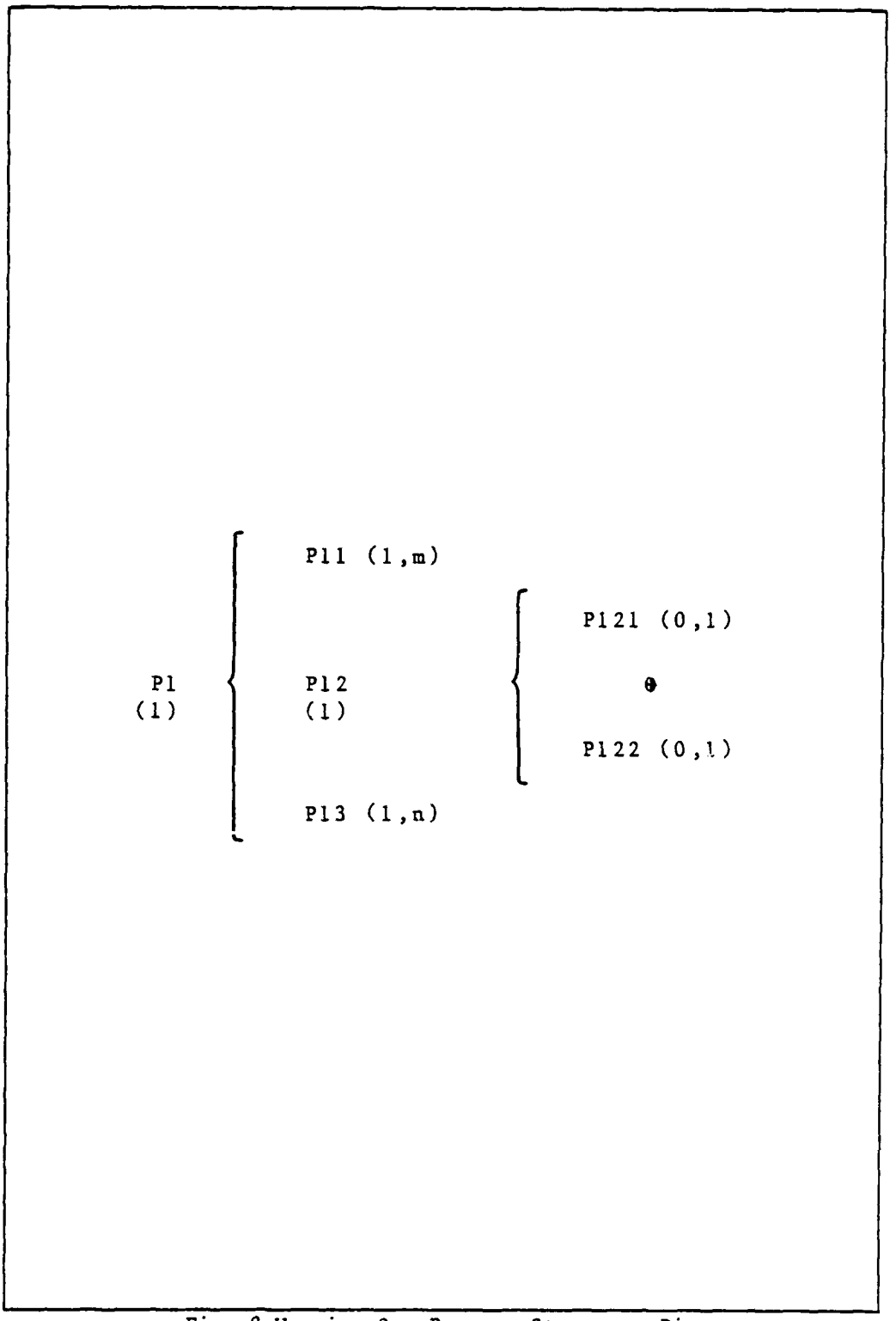



Fig. 8 Warnier-Orr Process Structure Diagram

Readability is of utmost importance to every activity following the design phase of software development. The design must be readable and understandable to those whose jobs will be to code, test and maintain the software.

The last advantage and one of the most important, practically, is the ease with which a Warnier-Orr design can be implemented. This applies to implementation in any computer language. The format of the Warnier-Orr diagram graphically represents the three constructs which are necessary and sufficient for any program implementation : sequence, decision and iteration (looping).

In Figure 8 module P1 consists of three sequentially (top to bottom) executed steps (sequence) : P11, P12 and P13. Module P12 consists of two steps : P121 and P122. At least one of P121 and P122, but not both, are executed each time P12 is executed (decision). The notations on P11 and P13 in parenthesis indicate how many times that module is repeated each time the parent module, P1, is executed. In the case of P11 and P13, the modules will be executed from one to "m" and one to "n" times, respectively. This represents looping. The notation on P1 indicates that the sequence P11-P12-P13 will be executed only once.

Extensions to the basic three constructs are obtained by combining them. A combination of the loop and decision constructs produces the DO-UNTIL or DO-WHILE constructs while multiple decisions illustrate the CASE statement of the PASCAL language. These extensions are mentioned to illustrate the relationships that the Warnier-Orr diagrams exhibit to implementation. This will be discussed in more detail in the following section.

The disadvantages of Warnier-Orr are not significant. These disadvantages refer to the flexibility in form allowed by the process. There are no set rules as to detailed conventions used on the diagrams (e.g.,whether zero to "n" repetitions is adopted as opposed to one to "n" repetitions). This leads to the necessity of rhetoric defining the conventions for different usages of the method.

Design System Choice

All three proposed alternative design methodologies have the same major goals : to make the design process logical within itself and to lead logically to an efficient implementation. Efficient, in this sense, is defined to include time of coding, time of execution and logical correctness (susceptibility to errors). Based on these factors, the Warnier-Orr approach will be used to design the software for the CORE graphics input system for the VAX 11/780 computer.

Whereas all three methodologies meet the goals listed, the Warnier-Orr technique is decidedly easier to read and exhibits, in its final form, a direct correlation to an implementation in any language. Warnier-Orr does not require the enormous amount of paperwork which necessarily accompanies the SADT method : such as data and process dictionaries. The definition of data and processes with Warnier-Orr is inherent in the diagrams.

Unlike Jackson's method, Warnier-Orr requires no listing of operations needed to perform the processes designed. These operations are more properly allocated to the implementation phase rather than to the design phase of software development. Warnier-Orr diagrams themselves provide direct indications as to what operations will be needed without the need of a separate listing.

The final reason for choosing the Warnier-Orr method is derived from a practical consideration related to the choice of software language to be used for implementation of the CORE standard.

Although implementation considerations theoretically do not or should not affect the design phase, the choice of a design methodology does have practical bearing on the implementation phase. The implementation of the CORE system discussed in this report is confined to a specific set of hardware and a specific operating environment (i.e., the Digital Equipment Corp. VAX 11/780 and the VMS operating system). This computer system supports the PASCAL language and a version of the FORTRAN-77 language. Unlike FORTRAN, PASCAL and the Warnier-Orr method are related on a one-to-one basis. For this reason it is strongly suggested that PASCAL be used for the actual implementation of the design for the CORE standard system.

## IV. Relating Design Requirements to System Capabilities

The purpose of this chapter is to discuss the interaction which occurs between the implemented software system and its working environment. These interactions require some consideration with regard to effects on the design requirements. In the following sections considerations involving the operating system environment, the implementation language and the graphics terminal are discussed.

### VAX/VMS Operating System Considerations

The VAX/VMS (Virtual Memory System) operating system is one which is both structured and flexible. It is structured in the sense that the command language is highly defined and flexible in that the system allows the user to determine at what level of interaction he or she wishes to incorporate the system's capabilities into his or her program. The user may incorporate the system's capabilities at the highest level of architecture and let the system itself take care of details. The user may also elect to handle certain details in his or her own applications program : to make the program more efficient, for example.

A clear example of this occurs in the input/output (I/O) processes. Within a user program, written in PASCAL for example, a simple WRITE statement may be sufficient to accomplish the output of data to a line printer or terminal. On the other hand, and at a lower system level, a call to a system utility routine may allow the user to perform block transfers of data from a buffer to the output device while the program continues on executing other statements.

It is this lower level of system hierarchy which provides the necessary capabilities needed to implement an asynchronous or level three CORE system. However, as is discussed later in this chapter, the

physical characteristics of the graphics terminal available for this system makes implementation of level three unneccessary. If, in the future, other graphics devices capable of asynchronous operation should be added to the system, the VAX/VMS operating system is capable of supporting level three CORE implementation.

Language Considerations

Although this design effort includes no attempt at source code implementation of the design, the choice of a programming language has some affect on the design technique used. For this reason, a discussion of the languages supported by the VAX/VMS operating system and the level of that support is appropriate and necessary.

The VAX/VMS operating system supports standard PASCAL with some non-standard variations. Further efforts to implement the CORE standard on the VAX 11/780 should use standard PASCAL. This recommendation is made for three reasons :

1. General structure - PASCAL is the most highly structured language supported by VAX/VMS and a highly structured language should accompany a highly structured design. This makes the design and implementation compatible.

2. Recursive procedures - Many of the routines in the CORE system are recursion oriented.

3. Design method similarity - The Warnier-Orr diagram looks very similar to a PASCAL program and is nearly one-for-one compatible with PASCAL statements.

Because of certain limitations inherent in the PASCAL language and the implementation of PASCAL on the VAX system, there is a possibility that some low level (transparent to the user) routines will need to be written in another language (e.g., FORTRAN or assembly language). If this becomes necessary the VAX version of PASCAL permits calls to separately compiled or assembled, external routines. These routines may be written in any of the other languages supported by the operating system. Writing support programs in languages other than PASCAL should be avoided, however, as the facility which provides for this within the PASCAL language is a nonstandard feature.

Graphics Terminal Considerations

The only graphics terminal available for consideration during the design phase was the Tektronix 4014. The major effect that this restriction has had in design is that only a level two CORE input system is necessary to support graphics displays on the 4014.

The 4014 terminal is incapable of performing multiple simultaneous input operations. Therefore, asynchronous (level three) implementation of the CORE standard would add nothing to a simple synchronous (level two) system.

As with any graphics terminal, the 4014 uses special character sequences to control the display process. Among these special sequences are included control characters (e.g., STX, ETX, FF, etc.). Normally, under high level I/O control (e.g., the WRITE and READ statements), control characters are reserved for operating system use only and, therefore, are not permitted to be transmitted between the program and I/O devices. However, under a lower level of I/O control (e.g., calls to the system utility SYS$QIO [Ref 4]), these characters are ignored by

the operating system and are thus transmitable characters.

The 4014 has two basic display modes : normal and write-through [Ref 3]. The normal mode causes the terminal to store all data on the terminal screen (storage screen). Whatever is drawn on the view screen remains until the entire screen is erased. In the write-through mode the screen is placed in refresh : the image drawn on the screen is not retained and must be continuously redrawn to remain visible. Refresh allows dynamic displays on the view screen provided the graphics software system continuously writes and updates the images.

However, refresh mode also introduces flicker to the display if the display is not drawn often enough. The refresh rate depends on two factors : the data transfer rate between program and the terminal and the total length of all vectors drawn on the screen. The length of vectors drawn on the screen can be affected very little other than by drawing physically smaller images on the screen.

The transfer rate, however, is affected by the speed at which the communication link between the computer and the terminal is operated. The 4014 terminal and the VAX/VMS operating system support a maximum data transfer rate of 9600 Baud. Therefore, high speed communication between the software driver and the graphics display is possible. This results in less flicker in the display, but does not entirely eliminate it.

One final constraint imposed on the design by the physical limitations of the 4014 hardware is one involving implementation of the STROKE class of devices. On the 4014 there is no way to implement this logical class of devices either explicitly or by simulation. There is no direct physical device available on the 4014 terminal with which to support a logical STROKE device. On some terminals the STROKE device is

simulated by a two or three-dimensional joy stick. However, on the 4014, the closest physical analog to a joy stick is the two dimensional thumbwheels and these are useless as a STROKE device simply because they do not provide enough freedom of movement across the view surface. Therefore, no STROKE device for the 4014 terminal will appear in this design.

NOTE

This design decision results in a modification to the minimum set of logical devices for which support by the CORE system will be supplied.

Comments

Theoretically, software design should be independent of system hardware and software capabilities and configurations. However, this implies that, in practice, parts of the design may not be realizable. The ultimate goal of any standard is to produce a totally independent system : independent in terms of programming language, operating system environment and hardware configuration. Implementation of the standard, on the other hand, on a specific system and in a specific language, must be both tailored to the environment in which it is used and faithful to the goal of supporting portability of user programs among different computer systems.

In the face of this dichotomy of purpose, the design of the CORE standard for use on the VAX computer, within the VMS operating system environment and utilizing the Tektronix 4014 graphics terminal actually becomes the implementation of the CORE system.

# V. Design of the CORE Input System

The actual software design of the CORE input system follows, basically, the outline given by sections 6.2.3 through 6.2.12 of the CORE standard [Ref 1 :II-74 through II-90]. The major exception is that the EVENT devices incorporated will operate synchronously. Therefore, sections 6.2.6 through 6.2.8 of the CORE standard are not applicable to this design.

The purpose of this chapter is to discuss the design in the areas of the CORE system data structure, device identification, device intitializing and enabling, device sampling, synchronous event handling, device echoing, setting device characteristics and device inquiry.

The data structure design is located in Appendix B and the process structure design is located in Appendix C.

## Data Structure Design and Device Identification

The CORE implementation data structure is simple and straightforward. The input system data structure consists of those elements whicn are necessary to describe the logical devices and to utilize their capabilites. The elements SEGMENT and NORMAL_COORD are not included. However, the input system will need access to this data to operate correctly. These data are properly part of the data structure of the output system of the CORE.

Device identification is accomplished via the first element of each device data structure. The DEVICE_NUMBER element of each data structure is self-explanatory and refers to the individual device within each class.

DEVICE_ECHO indicates the type of echo enabled for the specified device and the ECHO_POSITION element indicates the coordinate where the echo will occur.

DEVICE_INIT and DEVICE_ENABLE contain the values "initialized" or "not initialized" and "enabled" or "not enabled", respectively as the case may be.

The DEVICE_INFO element of the structure contains information pertaining to each particular device based on its DEVICE_CLASS.

It is intended that the design developed in this thesis be used in conjunction with a CORE output system design. It is not the purpose of this thesis to design such an output system. Therefore, the detail of the interplay between the two systems is intentionally omitted. Whenever the output system should be designed the issue of interaction will be resolved in detail.

Program Structure

The data structure and the input primitives required by the CORE standard define the program structure needed to complete the input system design.

All of the structured processes are FUNCTIONS (versus SUBROUTINES) in the sense that, upon completion of the process, a value is associated with the process name. This value is zero if no errors have occured during process invocation or the associated error number if one has occured. The errors returned for each primitive are listed in Appendix A.

Device Initializing and Enabling

Devices can be initialized and enabled in either of two ways and disabled in any of three ways. A single device can be initialized, enabled and disabled (INITIALIZE_DEVICE, ENABLE_DEVICE and DISABLE_DEVICE). A group of N devices within a specified CLASS may be initialized, enabled and disabled (INITIALIZE_GROUP, ENABLE_GROUP and DISABLE_GROUP). Finally all enabled devices can be disabled (DISABLE_ALL).

Devices can be TERMINATED (de-initialized) singly or in groups. The same conventions apply equally to the TERMINATE and INITIALIZE functions.

Sampling Devices

The two classes of SAMPLED devices (LOCATOR and VALUATOR) are sampled with READ primitives (READ_VALUATOR and READ_LOCATOR). If the specified LOCATOR or VALUATOR is not enabled, an error is returned. The current LOCATOR position is returned in the X and Y parameters of the READ_LOCATOR primitive and the current value of the VALUATOR is returned in the VALUE parameter of the READ_VALUATOR primitive.

Synchronous Input Functions

Due to the decision to design only a synchronous level for the CORE input system, event handling, device association and accessing event report data are omitted. These functions are fulfilled by synchronous routines that wait for operator action or a device time- out to occur before returning control to the application program.

AWAIT_ANY_BUTTON causes the system to wait until any BUTTON is pressed or until the specified TIME has elapsed. The BUTTON_NUM of the button which is pressed is returned.

AWAIT_PICK causes the system to wait until the device specified by PICK_NUM causes an event or until the TIME specified in the formal parameter list expires. The SEGMENT_NAME and PICK_ID of the primitive picked are returned.

AWAIT_KEYBOARD causes the system to wait until an event occurs (signified by a carriage return) from the KEYBOARD device specified by KEYBOARD_NUM or until the specified TIME has elapsed. If an event occurs, the INPUT_STRING and string LENGTH are returned.

The AWAIT_ANY_BUTTON_GET_LOCATOR and AWAIT_ANY_BUTTON_GET_VALUATOR functions cause the system to react exactly as with the AWAIT_ANY_BUTTON primitive with the added associations of the LOCATOR and VALUATOR devices, respectively. If the BUTTON event occurs, in addition to the data returned as has been defined, the current LOCATOR position and the current VALUATOR value are also returned, respectively.

Device Echoing

The SET_ECHO function identifies the ECHO_TYPE to be used for the specified device. These types are enumerated in Appendix A.

The SET_ECHO_GROUP function causes the ECHO_TYPE for a number of devices (N) within a DEVICE_CLASS to be set.

SET_ECHO_SURFACE causes the echo for the specified device to be performed on the specified SURFACE_NAME.

SET_ECHO_POSITION causes the echo reference position for the specified device to be set to ECHO_X, ECHO_Y.

Setting Device Characteristics

The functions in this group cause characteristics to be associated with the device specified, if possible, regardless of whether or not the device is enabled (the device must be initialized). The effect of the characteristic will be immediate if the device is enabled or will occur as soon as the device is enabled.

SET_PICK causes the APERTURE characteristic to be set for the specified device. APERTURE is specified in normalized device coordinates and its value denotes the length of a side of square centered at the position of the PICK device.

SET_KEYBOARD causes the values associated with the KEYBOARD_INFO element of the data structure for the specified device to be set. These values are BUFFER_SIZE, INITIAL_STRING and CURSOR_START.

The SET_BUTTON function causes the PROMPT_SWITCH for the specified device to be set. If PROMPT_SWITCH is zero, prompting for the device is turned off. If PROMPT_SWITCH is non-zero, prompting for the device is turned on. SET_ALL_BUTTONS accomplishes this function for all BUTTONS.

The SET_VALUATOR function causes the INITIAL_VALUE and the RANGE of the VALUATOR values to be set. The range is set by specifying a HIGH_VALUE and a LOW_VALUE. INITIAL_VALUE will set the VALUE element for the specified device. However, this VALUE will be immediately replaced with the actual value of the logical device.

Inquiry

The functions defined for this category provide the user application program with a means of determining the capabilities of the various input devices supported by the CORE input system.

INQUIRE_INPUT returns the system LEVEL ("synchronous" will always be returned), the DEVICE_COUNTS (an array with the number of available devices for each DEVICE_CLASS) and system TIMING (indicating whether or not time-outs for devices are supported and, if so, the reliability of the time-out support).

INQUIRE_LOCATOR_DIMENSION always returns the value "2D". This is only level supported by the Tektronix 4014 terminal LOCATOR (cross-hair).

INQUIRE_DEVICE_STATUS returns the values "intialized" or "not initialized" in the parameter INTIALIZED and "enabled" or "not enabled" in the parameter ENABLED.

The following functions return the specified values for the specified devices :

INQUIRE_ECHO.....ECHO_TYPE

INQUIRE_ECHO_SURFACE.....SURFACE_NAME

INQUIRE_ECHO_POSITION.....ECHO_X, ECHO_Y

INQUIRE_PICK.....PICK_INFO

INQUIRE_KEYBOARD.....KEYBOARD_INFO

INQUIRE_BUTTON.....BUTTON_INFO

INQUIRE_VALUATOR.....VALUATOR_INFO

Comments

The CORE input system design developed in this effort can be integrated with an output system design in order to obtain a complete CORE system.

Integration at the design level of developement is smooth in that the abstract function constructs, such as Generate_Cursor for example, have no set formulation. When dealing with coded routines, the flexibility in routine interaction is minimal or non-existant.

Once the designs are integrated only the most basic details are needed to turn this design into a coded system of routines.

## VI. Conclusions and Recommendations

The purpose of this investigation was to design a system of software routines to implement input device independence for graphic display using the ACM/SIGGRAPH CORE standard as the requirements document. This objective has been met using the Warnier-Orr technique of software design. During the design effort consideration was given to the specific graphics terminal to be used by the system (the Tektronix 4014) and the operating system environment in which the system will reside (VAX/VMS).

Conclusions drawn from and about the design as well as recommendations for follow-on efforts for the implementation and integration of the design are given in the following sections of this chapter.

### Design Conclusions

Much time and effort was spent analyzing the different techniques available to accomplish the design of the input CORE system. A technique was chosen (Warnier-Orr) based of its simplicity, readability and logical construction. Consideration was also given to how well the design techniques correlated to the high order language (PASCAL) available for implementation.

The requirements document (the CORE standard) was studied in depth to increase the author's understanding of what exactly was required to satisfy those requirements. There are problems as yet unresolved within the CORE standard which have hindered this investigation. Interaction with the ACM standards committee has not been possible. Therefore, those problems which have occured have been resolved by deletion from the system in most cases. For example, the CORE requires that at least

one STROKE device be supported at any level of implementation of the input system. This is neither physically nor logically possible with the Tektronix 4014 graphics terminal. Therefore, the design of a STROKE device was omitted.

Another example of a problem area leads to addition rather than deletion of system capabilites. This problem occurs in the synchronous input section and specifically refers to device associations. Device association provides for the acquisition of data from SAMPLED devices automatically whenever the "associated" EVENT device generates an event.

Association of SAMPLED devices with EVENT devices is provided by the CORE only between the BUTTON class and the LOCATOR and VALUATOR classes of devices. However, it is logical to expect that association of any SAMPLED device with any EVENT device would be both possible and desirable. For example, it might be necessary to determine the current position of the LOCATOR whenever a PICK occurs. Additions to the requirements of the standard were not made but are recommended for follow-on investigations.

The CORE standard also contains inconsistencies in the descriptions of the input primitives as concerns the primitives' error returns. As one example of these inconsistencies, error return number 603 ("DEVICE_CLASS is invalid") is listed for the primitive ENABLE_GROUP but not for ENABLE_DEVICE. It seems only reasonable that an error in the naming of the DEVICE_CLASS parameter can be made as easily in the invocation of one function as in another. Therefore, "DEVICE_CLASS is invalid" should be made a possible error return for every primitive which requires that parameter DEVICE_CLASS.

Despite these problems the design has proceeded smoothly. This has been due, in part, to the design technique employed and also to the VAX/VMS operating system flexibility.

Recommendations

It is hoped that follow-on effort will utilize the design developed during this investigation and actually implement the CORE input section on the VAX 11/780 computer/Tektronix 4014 combination.

It is recommended that the capabilities and facilities of the Tektronix PLOT10 graphics package be utilized as much as possible when coding the routines. This package already exists on the VAX computer in a library file (DMA0:[SYSMGR]PLOT10.OLB). These routines will greatly simplify the task of coding the software routines needed to implement the design of the CORE input system. Full advantage should be taken of this package in order to eliminate duplication of effort. The PASCAL "FORTRAN" qualifiers for procedure and function declarations [Ref 5 and 6] will allow these routines to be called from inside the PASCAL program (The PLOT10 routines were written in FORTRAN).

Implementation and integration of the full CORE standard in an interactive system should be the ultimate goal of follow-on efforts. implementation of the input section of the CORE as a next effort will carry the investigation a great distance toward that goal.

## Bibliography


1. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH", Computer Graphics, 13 (3) (August, 1979).

2. "Status REport of the Graphics Standards Planning Committee of ACM/SIGGRAPH", Computer Graphics, 13 (3) (Fall, 1977).

3. Tektronix 4014 and 4014-1 Computer Display Terminal Users Manual, Tektronix, Inc., (1974).

4. VAX/VMS System Services Reference Manual, Digital Equipment Corporation, March, 1980.

5. VAX-11 PASCAL Language Reference Manual, Digital Equipment Corporation, November, 1979.

6. VAX-11 PASCAL User's Guide, Digital Equipment Corporation, November, 1979.

7. Rowe,J., P. Keller and K. O'Hair,GRAFCORE Reference Manual, Lawrence Livermore Laboratory, March 1979 (UCID-30171).

8. Rowe,J., P. Keller and K. O'Hair,GRAFLIB - Useful Logic, Lawrence Livermore Laboratory, March 1979 (UCID-30172).

9. Jackson, Michael A., "Constructive Methods of Program Design", Proceedings, First Conference of the European Cooperation in Informatics, 44 : 394-412, 1976.

10. Orr, Kenneth T., "Introducing Structured Systems Design", Infotech State of the Art Report, Structured Analysis and Design, Infotech International Limited, Maidenhead, England.

11. Higgins, David A., "Structured Programming with Warnier-Orr Diagrams", Byte, 1978.

12. "An Introduction to SADT", SofTech, Incorporated, November 1976.

# Appendix A

This appendix contains a listing of the CORE input primitive functions, their associated formal parameter lists and the possible error returns for each function. Each primitive listing includes its reference paragraph for the 1979 CORE standard status report [Ref 1]. The listings are grouped into the nine categories listed in Chapter II of this text.

Initializing and Enabling Devices

[6.2.4.1] INITIALIZE_DEVICE (DEVICE_CLASS, DEVICE_NUM)

Errors:
601. The specified device is already initialized.
602. DEVICE_CLASS or DEVICE_NUM is invalid.

[6.2.4.2] INITIALIZE_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
604. One or more of the specified array entries is an invalid device number.

[6.2.4.3] ENABLE_DEVICE (DEVICE_CLASS, DEVICE_NUM)

Errors:
605. The specified device is not initialized.
606. The specified device is already enabled.

[6.2.4.4] ENABLE_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
607. One or more of the devices in the group is not initialized.

[6.2.4.5] DISABLE_DEVICE (DEVICE_CLASS, DEVICE_NUM)

608. The specified device is not enabled.

[6.2.4.6] DISABLE_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
607. One or more of the devices in the group is not initialized.

[6.2.4.7] DISABLE_ALL ()

Errors: None.

[6.2.4.8] TERMINATE_DEVICE (DEVICE_CLASS, DEVICE_NUM)

Errors:
605. The specified device is not initialized.

[6.2.4.9] TERMINATE_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
607. One or more of the devices in the group is not initialized.

## Reading Sampled Devices

[6.2.5.1] READ_LOCATOR_2 (LOCATOR_NUM, X, Y)
READ_LOCATOR_3 (LOCATOR_NUM, X, Y, Z)

Errors:
609. The specified LOCATOR device is not enabled.

[6.2.5.2] READ_VALUATOR (VALUATOR_NUM, VALUE)

Errors:
610. The specified VALUATOR device is not enabled.

Event Handling

[6.2.6.1] AWAIT_EVENT (TIME, EVENT_CLASS, EVENT_NUM)

Errors:
611. TIME is less than zero.

[6.2.6.2] FLUSH_DEVICE_EVENTS (EVENT_CLASS, EVENT_NUM)

Errors:
612. EVENT_CLASS and EVENT_NUM do not specify an initialized event device.

[6.2.6.3] FLUSH_GROUP_EVENTS (EVENT_CLASS, EVENT_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
607. One or more of the devices in the group is not initialized.
613. EVENT_CLASS is an invalid event device class.

[6.2.6.4] FLUSH_ALL_EVENTS ()

Errors: None.

Associating Devices

[6.2.7.1] ASSOCIATE (EVENT_CLASS, EVENT_NUM, SAMPLED_CLASS, SAMPLED_NUM)

Errors:
614. The specified association already exists.
615. EVENT_CLASS or SAMPLED_CLASS is an invalid class or a class of the wrong type.
616. One or both of the specified devices is not initialized.

[6.2.7.2] DISASSOCIATE (EVENT_CLASS, EVENT_NUM, SAMPLED_CLASS, SAMPLED_NUM)

Errors:
615. EVENT_CLASS or SAMPLED_CLASS is an invalid class or a class of the wrong type.
616. One or both of the specified devices is not initialized.
617. The specified association does not exist.

[6.2.7.3] DISASSOCIATE_DEVICE (DEVICE_CLASS, DEVICE_NUM)

Errors:
605. The specified device is not initialized.

[6.2.7.4] DISASSOCIATE_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
607. One or more of the devices in the group is not initialized.

[6.2.7.5] DISASSOCIATE_ALL ()

Errors: None.

Accessing Event Report Data

[6.2.8.1] GET_PICK_DATA (SEGMENT_NAME, PICK_ID)

Errors:
618. The current event report is not from a PICK device.

[6.2.8.2] GET_KEYBOARD_DATA (INPUT_STRING, NUM_INPUT)

Errors:
619. The current event report is not from a KEYBOARD device.

[6.2.8.3] GET_STROKE_DATA_2 (ARRAY_SIZE, X_ARRAY, Y_ARRAY, NUM_POSITIONS)
GET_STROKE_DATA_3 (ARRAY_SIZE_X_ARRAY, Y_ARRAY, Z_ARRAY)

Errors:
3. ARRAY_SIZE is less than or equal to zero.
620. The current event report is not from a STROKE device.
621. A 2D input function has been used when a 3D input function was needed to avoid information loss.

[6.2.8.4] GET_LOCATOR_DATA_2 (LOCATOR_NUM, X, Y)
GET_LOCATOR_DATA_3 (LOCATOR_NUM, X, Y, Z)

Errors:
621. A 2D function has been used when a 3D function was needed to avoid information loss.
622. When the event occurred, the specified LOCATOR device was not enabled or was not associated with the event device that caused the current event report.

[6.2.8.5] GET_VALUATOR_DATA (VALUATOR_NUM, VALUE)

Errors:
623. When the event occurred, the specified VALUATOR device was not enabled or was not associated with the event device that caused the current event report.

Synchronous Input

[6.2.9.1] AWAIT_ANY_BUTTON (TIME, BUTTON_NUM)

Errors:
611. TIME is less than zero.
624. No BUTTON devices are initialized.

[6.2.9.2] AWAIT_PICK (TIME, PICK_NUM, SEGMENT_NAME, PICK_ID)

Errors:
611. TIME is less than zero.
625. The specified PICK device is not initialized.

[6.2.9.3] AWAIT_KEYBOARD (TIME, KEYBOARD_NUM, INPUT_STRING, LENGTH)

Errors:
611. TIME is less than zero.
626. The specified KEYBOARD device is not initialized

[6.2.9.4] AWAIT_STROKE_2 (TIME, STROKE_NUM, ARRAY_SIZE,
X_ARRAY, Y_ARRAY, NUM_POSITIONS)
AWAIT_STROKE_3 (TIME, STROKE_NUM, ARRAY_SIZE, X_ARRAY,
Y_ARRAY, Z_ARRAY, NUM_POSITIONS)

Errors:
3. ARRAY_SIZE is less than or equal to zero.
611. TIME is less than zero.
621. A 2D input function was used when a 3D input function was needed to avoid information loss.
627. The specified STROKE device is not initialized.

[6.2.9.5] AWAIT_ANY_BUTTON_GET_LOCATOR_2 (TIME, LOCATOR_NUM,
BUTTON_NUM, X, Y)
AWAIT_ANY_BUTTON_GET_LOCATOR_3 (TIME, LOCATOR_NUM,
BUTTON_NUM, X, Y, Z)

Errors:
611. TIME is less than zero.
621. A 2D input function was used when a 3D input function was needed to avoid information loss.
624. No BUTTON devices are initialized.
628. The specified LOCATOR device is not initialized.

[6.2.9.6] AWAIT_ANY_BUTTON_GET_VALUATOR (TIME, VALUATOR_NUM, BUTTON_NUM, VALUE)

Errors:
611. TIME is less than zero.
624. No BUTTON devices are initialized.
629. The specified VALUATOR device is not intialized.

Device Echoing

[6.2.10.1] SET_ECHO (DEVICE_CLASS, DEVICE_NUM, ECHO_TYPE)

Errors:
605. The specified device is not initialized.
630. The specified echo type is invalid for the specified device class.

[6.2.10.2] SET_ECHO_GROUP (DEVICE_CLASS, DEVICE_NUM_ARRAY, N, ECHO_TYPE)

Errors:
2. N is less than or equal to zero.
603. DEVICE_CLASS is invalid.
607. One or more of the devices in the group is not initialized.
630. The specified echo type is invalid for the specified device class.

[6.2.10.3] SET_ECHO_SEGMENT (DEVICE_CLASS, DEVICE_NUM, SEGMENT_NAME)

Errors:
305. There is no retained segment named SEGMENT_NAME.
605. The specified device is not initialized.
631. The specified device does not have an ECHO_TYPE that requires an echo segment.
632. The IMAGE_TRANSFORMATION_TYPE of the specified segment is incompatible with the current ECHO_TYPE.

[6.2.10.4] SET_ECHO_SURFACE (DEVICE_CLASS, DEVICE_NUM, SURFACE_NAME)

Errors:
605. The specified device is not intialized.
708. The specified view surface is not initialized.

[6.2.10.5] SET_ECHO_POSITION (DEVICE_CLASS, DEVICE_NUM, ECHO_X, ECHOY)

Errors:
605. The specified device is not initialized.
633. ECHO_X and ECHO_Y specify a position outside the normalized device coordinate space.

Setting Input Device Characteristics

[6.2.11.1] SET_PICK (PICK_NUM, APERTURE)

Errors:
625. The specified PICK device is not initialized.
634. APERTURE is less than or equal to zero.

[6.2.11.2] SET_KEYBOARD (KEYBOARD_NUM, BUFFER_SIZE, INITIAL_STRING, CURSOR_START)

Errors:
626. The specified KEYBOARD device is not initialized
635. BUFFER_SIZE is less than or equal to zero or greater than the implementation-defined maximum.
636. CURSOR_START is not greater than zero and less than or equal to BUFFER_SIZE.
637. INITIAL_STRING contains one or more undefined characters.

[6.2.11.3] SET_BUTTON (BUTTON_NUM, PROMPT_SWITCH)

Errors:
638. The specified BUTTON device is not initialized.

[6.2.11.4] SET_ALL_BUTTONS (PROMPT_SWITCH)

Errors:
624. No BUTTON devices are initialized.

[6.2.11.5] SET_STROKE (STROKE_NUM, BUFFER_SIZE, DISTANCE, TIME)

Errors:
627. The specified STROKE device is not initialized.
635. BUFFER_SIZE is less than or equal to zero or greater than the implementation-defined maximum.
639. DISTANCE or TIME is less than or equal to zero.

[6.2.11.6] SET_LOCATOR_2 (LOCATOR_NUM, LOC_X, LOC_Y)
SET_LOCATOR_3 (LOCATOR_NUM, LOC_X, LOC_Y, LOC_Z)

Errors:
628. The specified LOCATOR device is not initialized.
640. LOC_X and LOC_Y (and LOC_Z) specify a position outside normalized device coordinate space.

[6.2.11.7] SET_LOCPORT_2 (LOCATOR_NUM, XMIN, XMAX, YMIN, YMAX)
SET_LOCPORT_3 (LOCATOR_NUM, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

Errors:
628. The specified LOCATOR device is not initialized.
641. XMIN is not less than XMAX, or YMIN is not less than YMAX (or ZMIN is not less than ZMAX).
642. One or more of the locport corners is outside the normalized device coordinate space.

[6.2.11.8] SET_VALUATOR (VALUATOR_NUM, INITIAL_VALUE, LOW_VALUE, HIGH_VALUE)

Errors:
629. The specified VALUATOR device is not intialized.
643. LOW_VALUE is greater than HIGH_VALUE.
644. INITIAL_VALUE does not lie in the range defined by LOW_VALUE and HIGH_VALUE.

Inquiry

[6.2.12.1] INQUIRE_INPUT_CAPABILITIES (LEVEL, DEVICE_COUNTS, TIMING)

Errors: None.

[6.2.12.2] INQUIRE_INPUT_DEVICE_CHARACTERISTICS (DEVICE_CLASS, DEVICE_NUM, IMPLEMENTATION, ECHO, VIEW_SURFACE, ASSOCIATION_SIZE, ASSOCIATION_CLASS, ASSOCIATION_NUM, ASSOCIATION_COUNT, DUPLICATION_SIZE, DUPLICATION_CLASS, DUPLICATION_NUM, DUPLICATION_COUNT, PRECISION, DELAY)

Errors:
605. The specified device is not initialized.
645. ASSOCIATION_SIZE is less than or equal to zero.
646. DUPLICATION_SIZE is less than or equal to zero.

[6.2.12.3] INQUIRE_STROKE_DIMENSION (STROKE_NUM, DIMENSION)
INQUIRE_LOCATOR_DIMENSION (LOCATOR_NUM, DIMENSION)

Errors:
605. The specified device is not initialized.

[6.2.12.4] INQUIRE_DEVICE_STATUS (DEVICE_CLASS, DEVICE_NUM, INITIALIZED, ENABLED)

Errors:
602. DEVICE_CLASS or DEVICE_NUM is invalid.

[6.2.12.5] INQUIRE_DEVICE_ASSOCIATIONS (EVENT_CLASS, EVENT_NUM, ARRAY_SIZE, SAMPLED_CLASS_ARRAY, SAMPLED_NUM_ARRAY, NUMBER_OF_ASSOCIATIONS)

Errors:
3. ARRAY_SIZE is less than or equal to zero.
612. EVENT_CLASS and EVENT_NUM do not specify an initialized event device.

[6.2.12.6] INQUIRE_ECHO (DEVICE_CLASS, DEVICE_NUM, ECHO_TYPE)
INQUIRE_ECHO_SURFACE (DEVICE_CLASS, DEVICE_NUM, SURFACE_NAME)
INQUIRE_ECHO_POSITION (DEVICE_CLASS, DEVICE_NUM, ECHO_X, ECHO_Y)
INQUIRE_PICK (PICK_NUM, APERTURE)
INQUIRE_KEYBOARD (KEYBOARD_NUM, BUFFER_SIZE, INITIAL_STRING, CURSOR_START)
INQUIRE_BUTTON (BUTTON_NUM, PROMPT_SWITCH)
INQUIRE_STROKE (STROKE_NUM, BUFFER_SIZE, DISTANCE, TIME)
INQUIRE_LOCATOR_2 (LOCATOR_NUM, LOC_X, LOC_Y)
INQUIRE_LOCATOR_3 (LOCATOR_NUM, LOC_X, LOC_Y, LOC_Z)
INQUIRE_LOCPORT_2 (LOCATOR_NUM, XMIN, XMAX, YMIN, YMAX)

```
INQUIRE_LOCPORT_3  (LOCATOR_NUM, XMIN, XMAX, YMIN, YMAX,
  ZMIN, ZMAX)
INQUIRE_VALUATOR  (VALUATOR_NUM, INITIAL_VALUE, LOW_VALUE,
  HIGH_VALUE)
```

Errors:

1. A 2D inquiry function has been used when a 3d inquiry function was needed tc avoid information loss.

605. The specified device is not initialized.

[6.2.12.7] `INQUIRE_ECHO_SEGMENTS  (DEVICE_CLASS, DEVICE_NUM, ARRAY_SIZE, ECHO_SEGMENT_ARRAY, NUMBER_ECHO_SEGMENTS)`

Errors:

3. ARRAY_SIZE is less than or equal to zero.

605. The specified device is not initialized.

## Appendix B

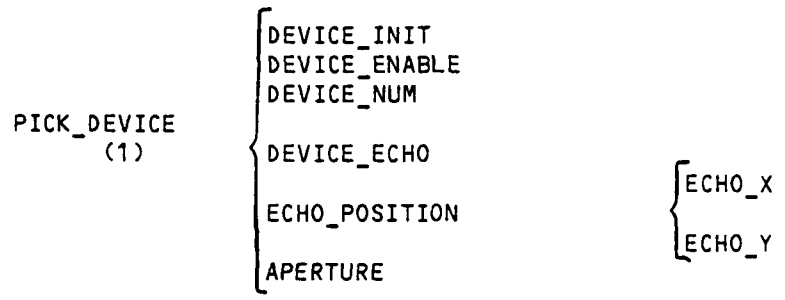
This appendix contains the Warnier-Orr data structure diagrams for the CORE input system. Each record represents a DEVICE_CLASS collection. When INITIALIZED_CORE is called (this must be the first call to CORE) all values in the data structure are initialized as follows :

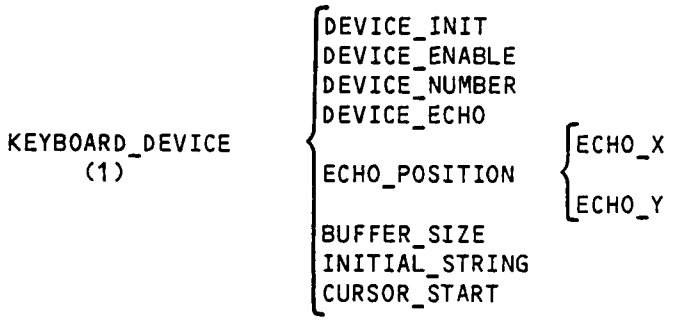
General :

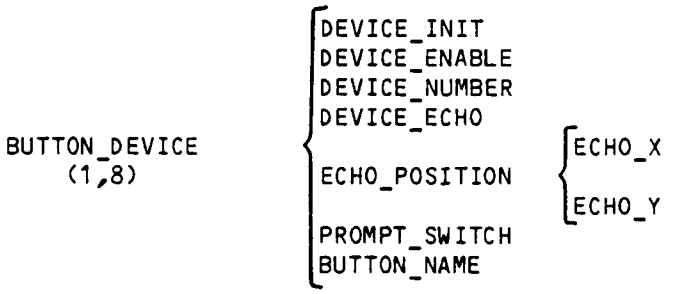
```
DEVICE_INIT = 'not initialized'
DEVICE_ENABLED = 'disabled'
DEVICE_NUMBER = 0
DEVICE_ECHO = 0
ECHO_POSITION = 0 , 0
```

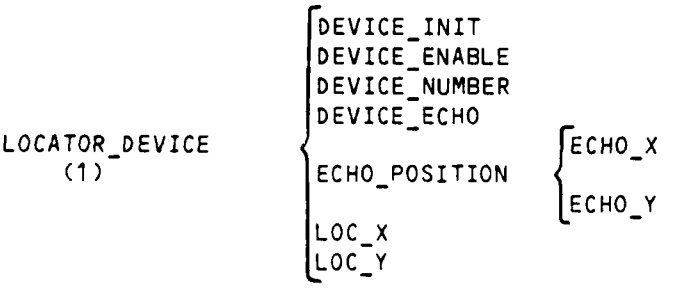
Specific :

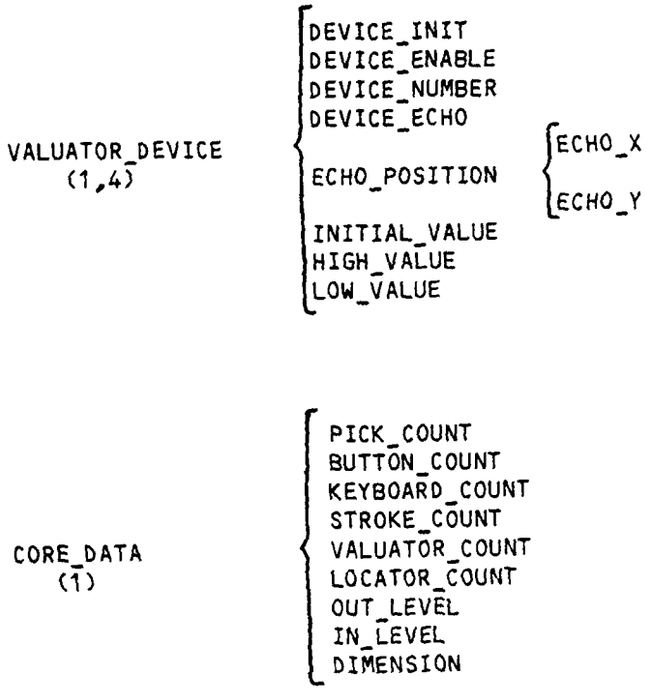
```
PICK_APERTURE = 1
KEYBOARD.BUFFER_SIZE = 80
KEYBOARD.INITIAL_STRING =  null
KEYBOARD.CURSOR_START = 1

BUTTON.PROMPT_SWITCH = 0
BUTTON.NAME =  blank

LOCATOR.LOC_X = 0
LOCATOR.LOC_Y = 0

VALUATOR.INITIAL_VALUE = 0
VALUATOR.HIGH_VALUE = 1
VALUATOR.LOW_VALUATOR = 0
```


PICK_DEVICE
(1)
DEVICE_INIT
DEVICE_ENABLE
DEVICE_NUM
DEVICE_ECHO
ECHO_POSITION
ECHO_X
ECHO_Y
APERTURE


KEYBOARD_DEVICE
(1)
DEVICE_INIT
DEVICE_ENABLE
DEVICE_NUMBER
DEVICE_ECHO
ECHO_POSITION
ECHO_X
ECHO_Y
BUFFER_SIZE
INITIAL_STRING
CURSOR_START


BUTTON_DEVICE
(1,8)
DEVICE_INIT
DEVICE_ENABLE
DEVICE_NUMBER
DEVICE_ECHO
ECHO_POSITION
ECHO_X
ECHO_Y
PROMPT_SWITCH
BUTTON_NAME


LOCATOR_DEVICE
(1)
DEVICE_INIT
DEVICE_ENABLE
DEVICE_NUMBER
DEVICE_ECHO
ECHO_POSITION
ECHO_X
ECHO_Y
LOC_X
LOC_Y


VALUATOR_DEVICE
(1,4)
DEVICE_INIT
DEVICE_ENABLE
DEVICE_NUMBER
DEVICE_ECHO
ECHO_POSITION
ECHO_X
ECHO_Y
INITIAL_VALUE
HIGH_VALUE
LOW_VALUE


CORE_DATA
(1)
PICK_COUNT
BUTTON_COUNT
KEYBOARD_COUNT
STROKE_COUNT
VALUATOR_COUNT
LOCATOR_COUNT
OUT_LEVEL
IN_LEVEL
DIMENSION

## Appendix C

This appendix contains the CORE process structure Warnier-Orr diagrams. The process structure is complete as far as the set of input primitive functions is concerned. Every input function is explicitly represented.

The local conventions used in these diagrams were :

1. The repeat factor usually appearing in parenthesis with the process name has been omitted if it is one (1). If a process is repeated (e.g., (1,N)) then that is explicitly stated.

2. The diagrams relate to the function calls listed in Appendix A of this document. All parameter lists for the primitives are located in that appendix and are not repeated with the process diagrams. The parameters listed in the function calls are assumed to be available within the process structure diagram or passed explicitly to lower level routines. Any other data needed within the diagram is provided explicitly.

3. When reference is made to fields of a record, a period is used to separate them. An example would be when referencing the ECHO_X field of the BUTTON_DEVICE the representation of that field in the diagrams would be :

BUTTON_DEVICE.ECHO_POSITION.ECHO_X

This is compatible with the standard PASCAL data record reference.

4. Whenever the CLASS of a device is obvious (e.g., stated in the function name) it is omitted from the reference of a data field. An example of this is the READ_LOCATOR_2 primitive. It is obvious that the primitive references LOCATOR devices. Therefore, the LOCATOR_DEVICE tag is omitted from references to that data record.

5. When reference to a specific DEVICE_NUM within a CLASS is required, the DEVICE_NUM appears in brackets. The following is an example from the INQUIRE_ECHO_POSITION primitive :

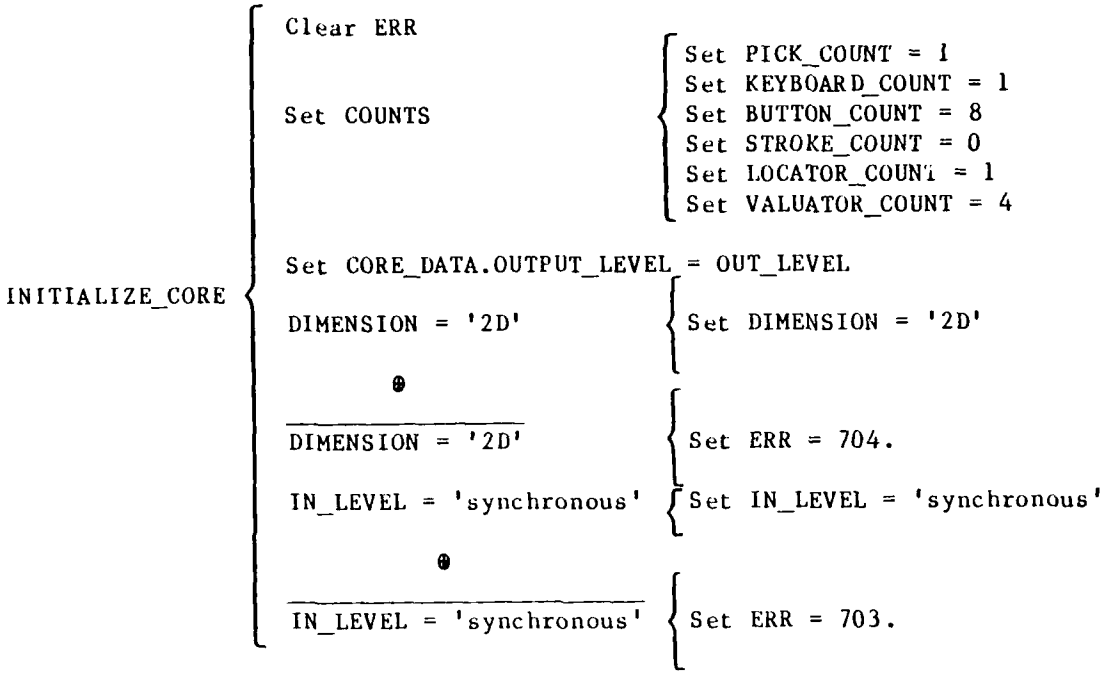
   Set ECHO_X = DEVICE_CLASS[DEVICE_NUM].ECHO_POSITION.ECHO_X

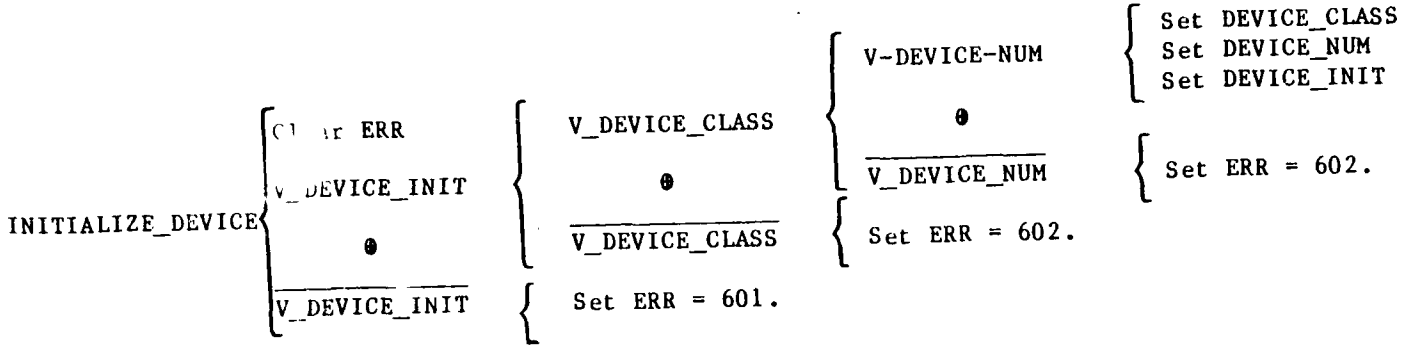
6. When two device classes appear in the same primitive parameter list, within that primitive any reference to either of those classes is made explicitly in order to avoid confusion. Such as :

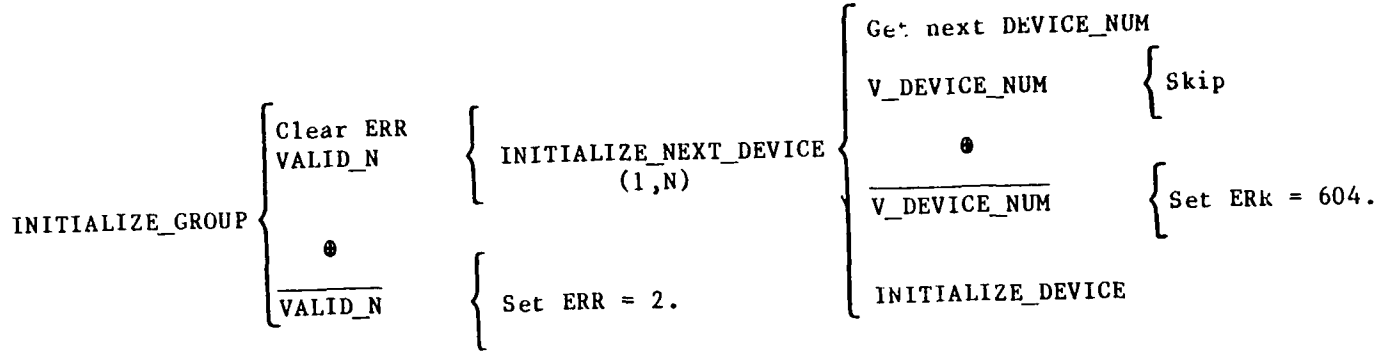
   AWAIT_ANY_BUTTON_GET_LOCATOR_2
   (...,LOCATOR_NUM, BUTTON_NUM,...,...)

7. The routine "SKIP" appearing in the diagrams is a "do nothing" routine and consists simply of a return to the

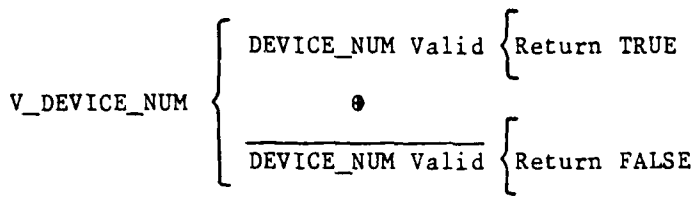
calling routine.


INITIALIZE_CORE
Clear ERR
Set COUNTS
Set PICK_COUNT = 1
Set KEYBOARD_COUNT = 1
Set BUTTON_COUNT = 8
Set STROKE_COUNT = 0
Set LOCATOR_COUNT = 1
Set VALUATOR_COUNT = 4
Set CORE_DATA.OUTPUT_LEVEL = OUT_LEVEL
DIMENSION = '2D'
Set DIMENSION = '2D'
DIMENSION = '2D'
Set ERR = 704.
IN_LEVEL = 'synchronous'
Set IN_LEVEL = 'synchronous'
IN_LEVEL = 'synchronous'
Set ERR = 703.

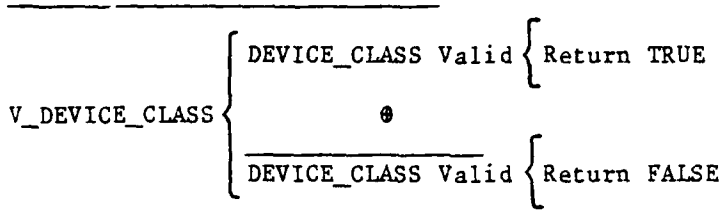

INITIALIZE_DEVICE
Clear ERR
V_DEVICE_INIT
●
V_DEVICE_INIT
V_DEVICE_CLASS
●
V_DEVICE_CLASS
Set ERR = 601.
V_DEVICE_CLASS
●
V_DEVICE_CLASS
Set ERR = 602.
V-DEVICE-NUM
●
V_DEVICE_NUM
Set DEVICE_CLASS
Set DEVICE_NUM
Set DEVICE_INIT
Set ERR = 602.


INITIALIZE_GROUP
Clear ERR
VALID_N
●
VALID_N
Set ERR = 2.
INITIALIZE_NEXT_DEVICE
(1,N)
Get next DEVICE_NUM
V_DEVICE_NUM
●
V_DEVICE_NUM
INITIALIZE_DEVICE
Skip
Set ERR = 604.

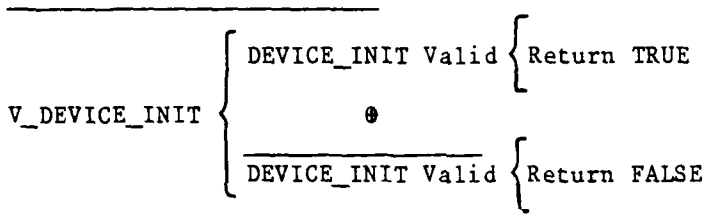
V_DEVICE_NUM (DEVICE_NUM)



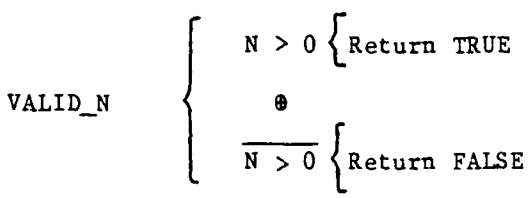

V_DEVICE_CLASS (DEVICE_CLASS)



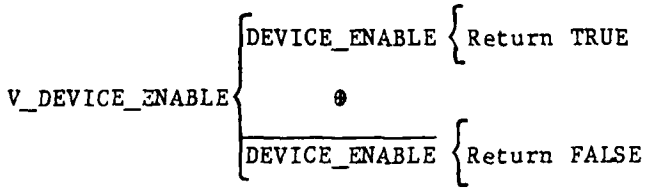

V_DEVICE_INIT (DEVICE_CLASS, DEVICE_NUM)




VALID_N (N)




V_DEVICE_ENABLE (DEVICE_CLASS, DEVICE_NUM)



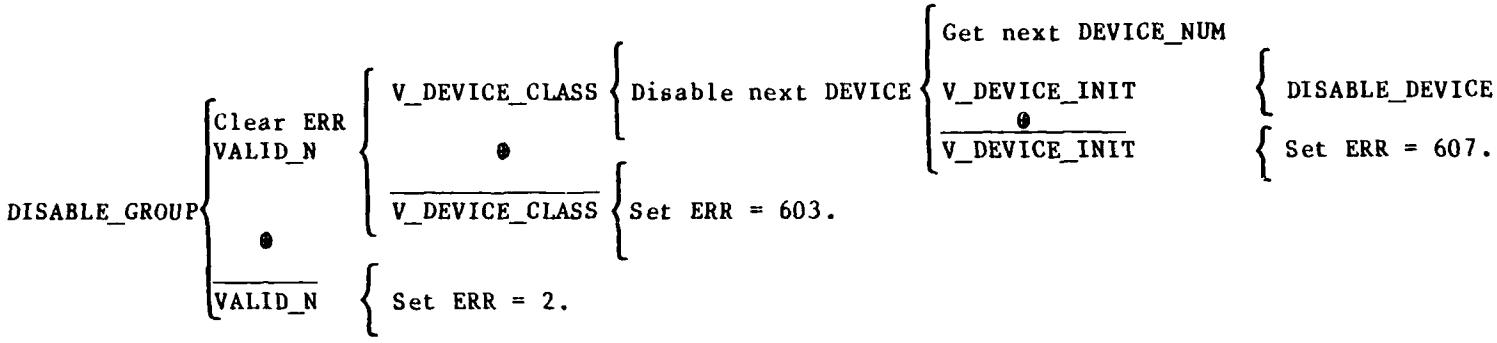
```
                    ⎧ Clear ERR
                    ⎪
                    ⎪ V_DEVICE_ENABLE ⎧ Set DEVICE_ENABLE
DISABLE_DEVICE      ⎨                 ⎩
                    ⎪       ⊕
                    ⎪
                    ⎪ V_DEVICE_ENABLE ⎧ Set ERR = 608.
                    ⎩                 ⎩
```

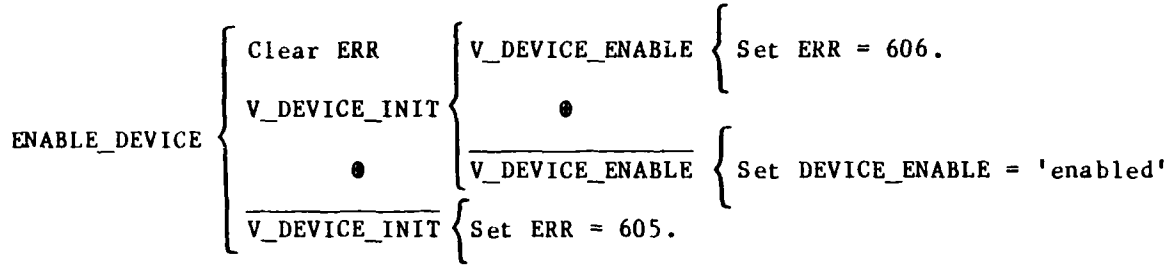
```
                    ⎧ Clear ERR
                    ⎪
                    ⎪ DISABLE_DEVICE ('PICK', PICK_DEVICE[PICK_NUM])
                    ⎪
                    ⎪ DISABLE_DEVICE ('KEYBOARD, KEYBOARD_DEVICE[KEYBOARD_NUM])
DISABLE_ALL         ⎨ DISABLE_DEVICE ('BUTTON', N)
                    ⎪      (1,8)
                    ⎪
                    ⎪ DISABLE_DEVICE ('LOCATOR', LOCATOR_DEVICE[LOCATOR_NUM])
                    ⎪
                    ⎪ DISABLE_DEVICE ('VALUATOR', N)
                    ⎩      (1,4)
```

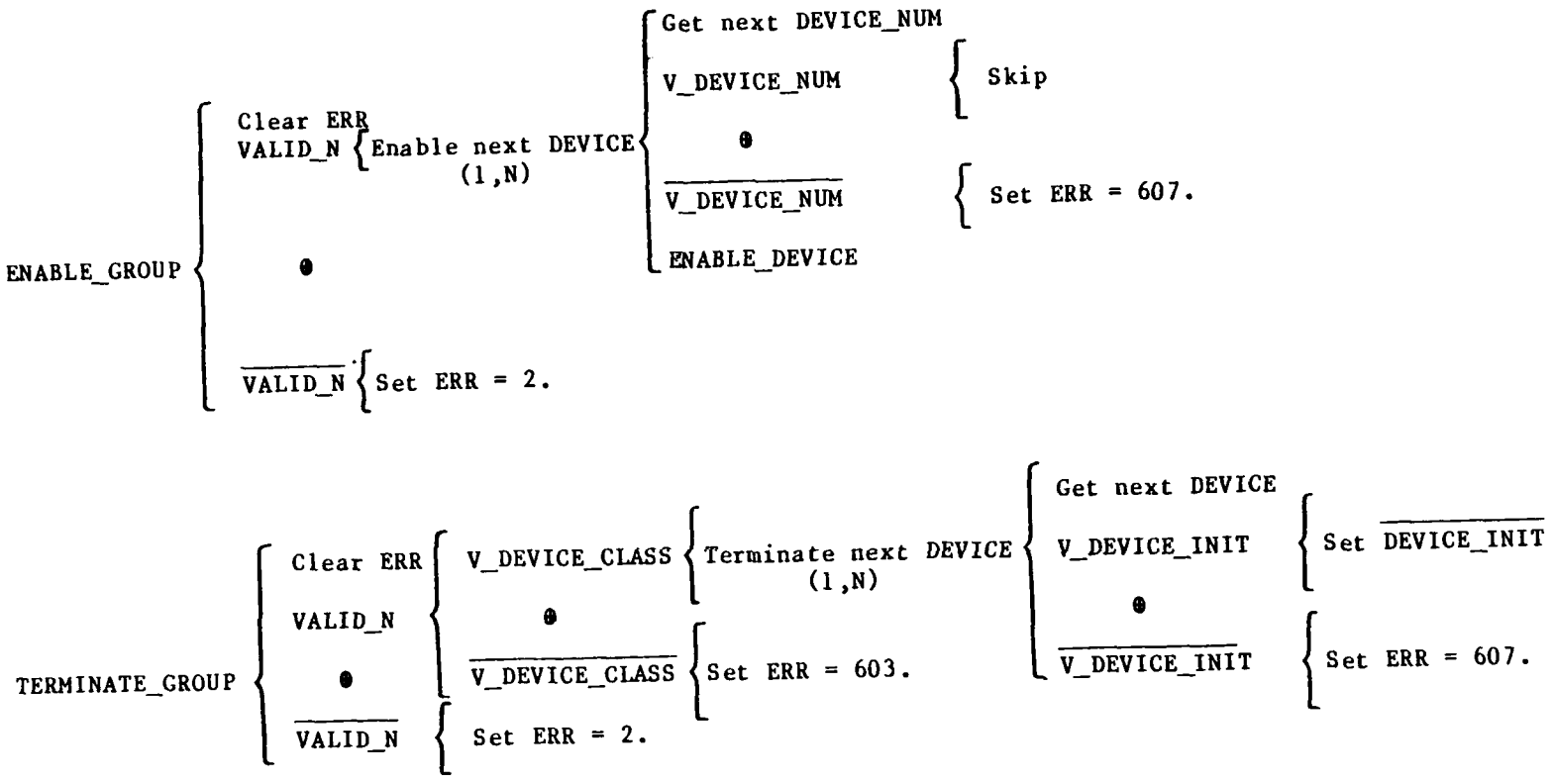
```
                    ⎧ Clear ERR
                    ⎪
                    ⎪ V_DEVICE_INIT ⎧ Set DEVICE_INIT
TERMINATE_DEVICE    ⎨               ⎩
                    ⎪       ⊕
                    ⎪
                    ⎪ V_DEVICE_INIT ⎧ Set ERR = 605.
                    ⎩               ⎩
```

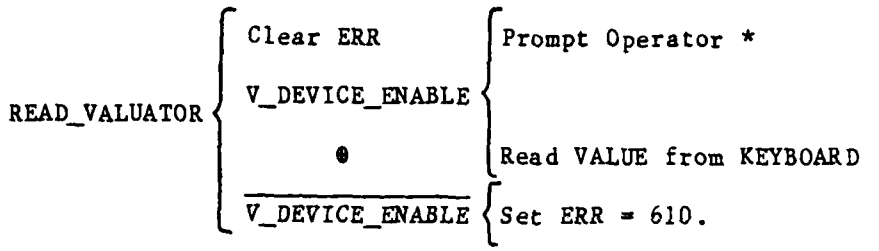
```
                    ⎧ Clear ERR
                    ⎪                 ⎧ Command Cursor to 4014  *
                    ⎪ V_DEVICE_ENABLE ⎨ Read X from 4014
READ_LOCATOR_2      ⎨                 ⎩ Read Y from 4014
                    ⎪       ⊕
                    ⎪
                    ⎪ V_DEVICE_ENABLE ⎧ Set ERR = 609.
                    ⎩                 ⎩
```

* This consists of sending two control characters to the 4014 terminal. The terminal responds with X and Y values corresponding to the position of the cursor (LOCATOR)


DISABLE_GROUP
Clear ERR
VALID_N
V_DEVICE_CLASS
V_DEVICE_CLASS
VALID_N
Set ERR = 2.
Set ERR = 603.
Disable next DEVICE
Get next DEVICE_NUM
V_DEVICE_INIT
V_DEVICE_INIT
DISABLE_DEVICE
Set ERR = 607.

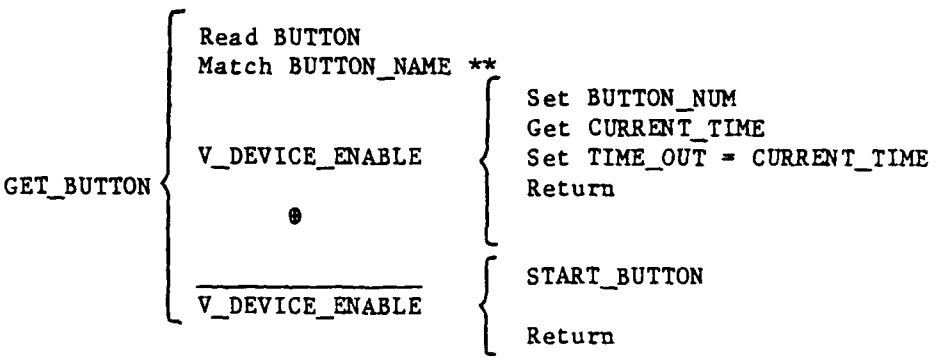

ENABLE_DEVICE
Clear ERR
V_DEVICE_INIT
V_DEVICE_INIT
V_DEVICE_ENABLE
V_DEVICE_ENABLE
Set ERR = 606.
Set DEVICE_ENABLE = 'enabled'
Set ERR = 605.


ENABLE_GROUP
Clear ERR
VALID_N
Enable next DEVICE (1,N)
Get next DEVICE_NUM
V_DEVICE_NUM
Skip
V_DEVICE_NUM
Set ERR = 607.
ENABLE_DEVICE
VALID_N
Set ERR = 2.
TERMINATE_GROUP
Clear ERR
VALID_N
V_DEVICE_CLASS
Terminate next DEVICE (1,N)
Get next DEVICE
V_DEVICE_INIT
Set DEVICE_INIT
V_DEVICE_INIT
Set ERR = 607.
V_DEVICE_CLASS
Set ERR = 603.
VALID_N
Set ERR = 2.

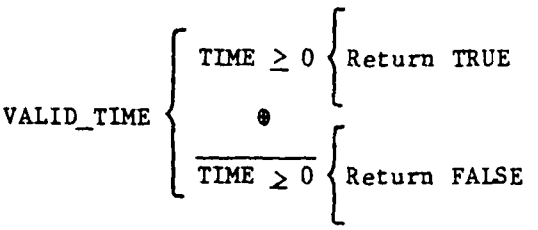



* This is a string prompt directing Operator to enter VALUE from KEYBOARD.

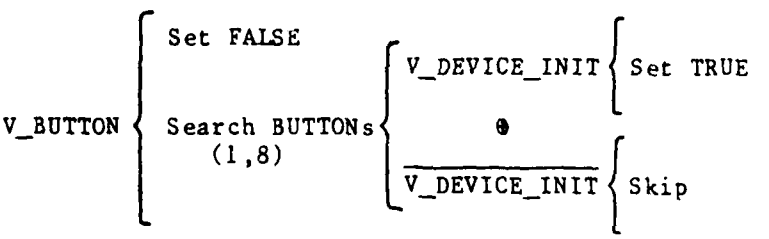
GET_BUTTON (BUTTON_NUM, TIME_OUT)

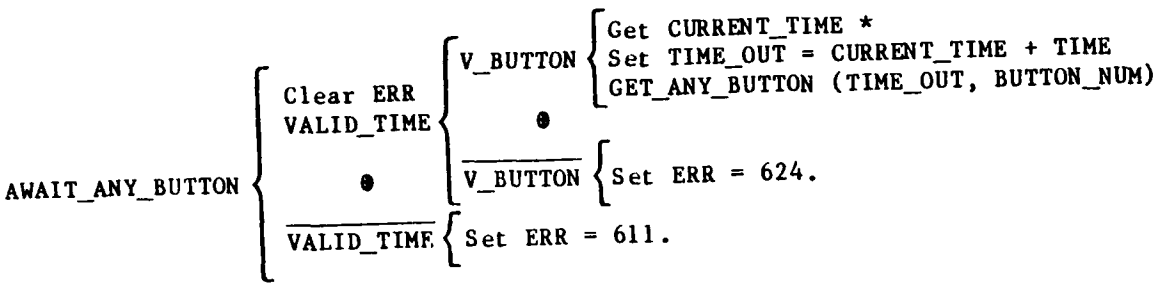



VALID_TIME (TIME)

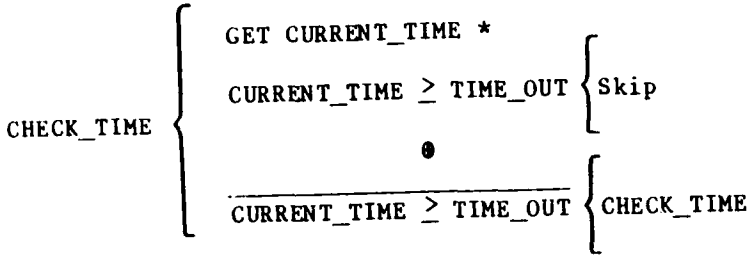



V_BUTTON ()

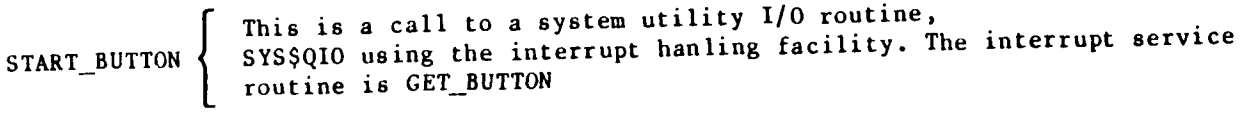




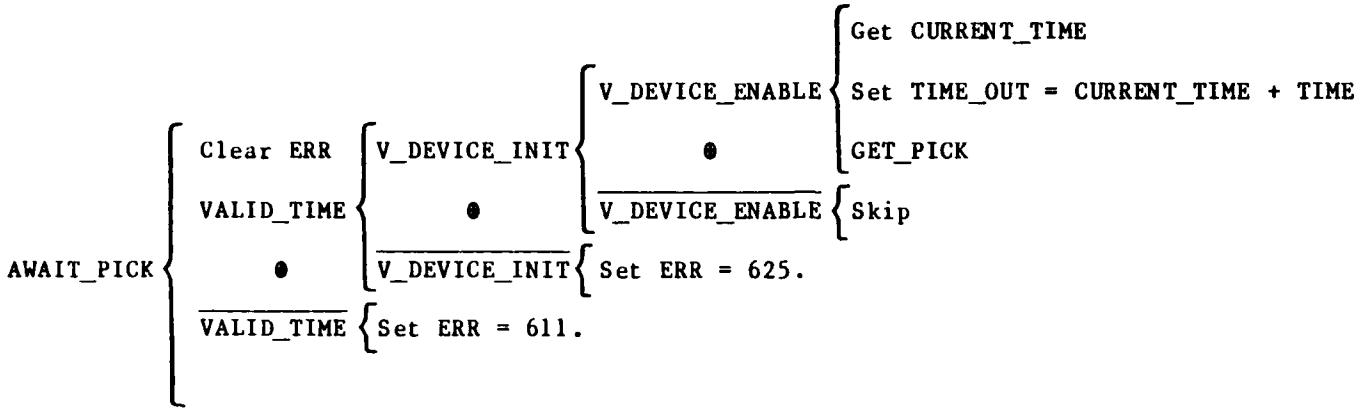

GET_ANY_BUTTON (TIME_OUT, BUTTON_NUM)

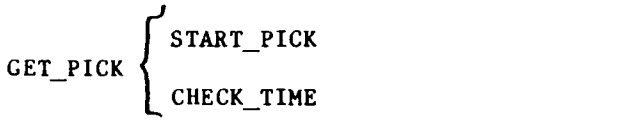

START_BUTTON (BUTTON, TIME_OUT)



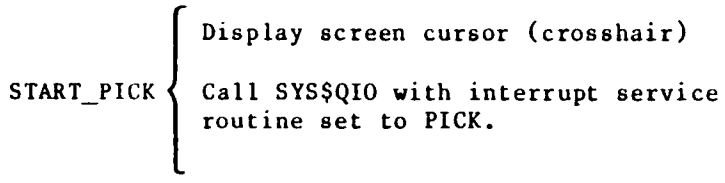

* From system utility routine 'TIME'




GET_PICK (SEGMENT_NAME, PICK_ID, TIME_OUT)




START_PICK



PICK (SEGMENT_NAME, PICK_ID, TIME_OUT)

```
PICK { Read any KEYBOARD key (any character)
       Read X (generated by terminal)
       Read Y (generated by terminal)
       GET_SEGMANT_NAME *
       GET_PICK_ID *
       GET_CURRENT_TIME
       Set TIME_OUT = CURRENT_TIME
       Return
```

* These two routines will be found in the OUTPUT section of the CORE standard and will map coordinate position (X,Y) into a SEGMENT and PRIMITIVE. For completness in this design they return 0.

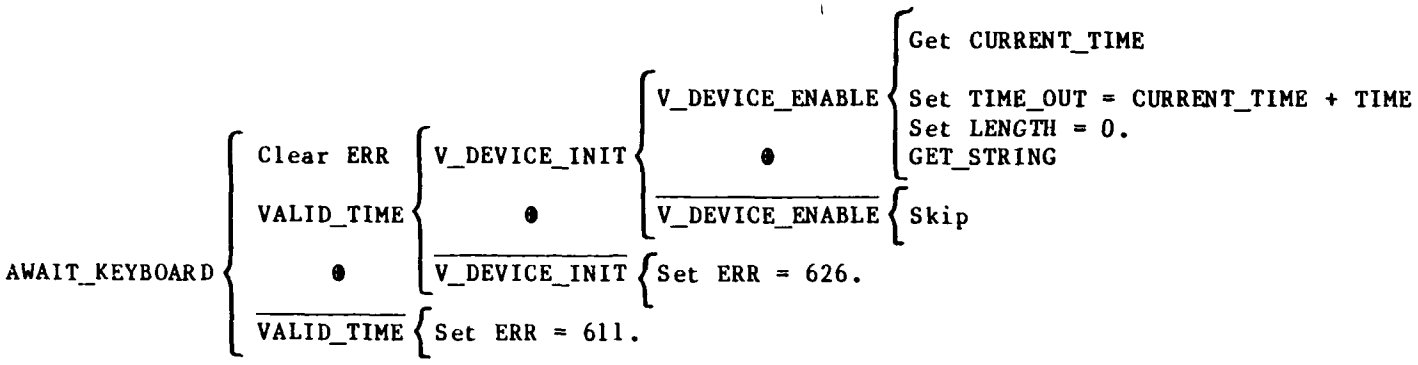
KEYBOARD (INPUT_STRING, LENGTH, TIME_OUT)

```
KEYBOARD { Read character from KEYBOARD (terminal)
           CHARACTER = CR* { Get CURRENT_TIME
                            Set TIME_OUT = CURRENT_TIME
                            Return
           •
           CHARACTER = CR { Update LENGTH by 1
                            Put CHARACTER into INPUT_STRING
                            GET_STRING
                            Return
```

* Carriage Return

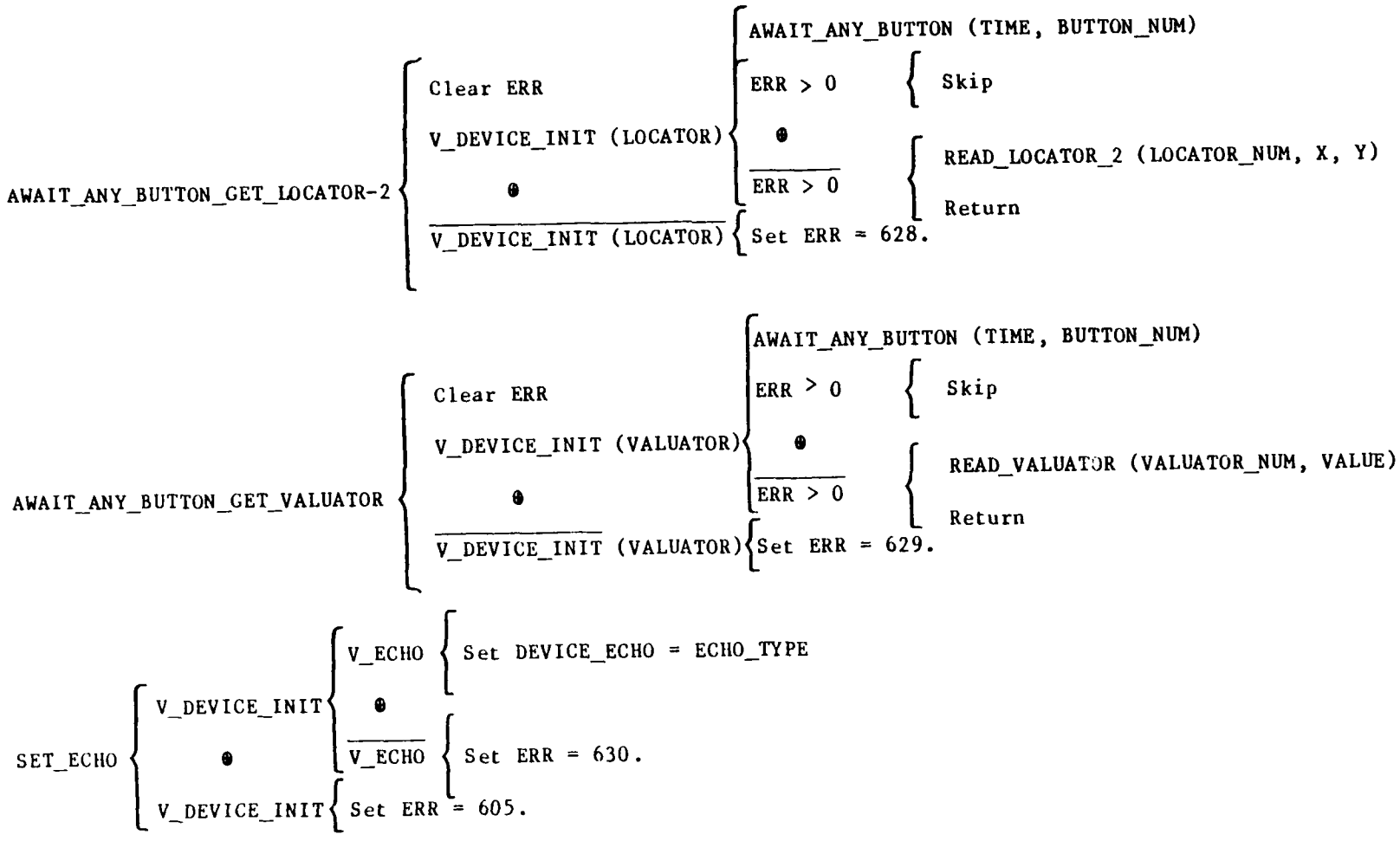
V_ECHO (DEVICE_CLASS, ECHO_TYPE)

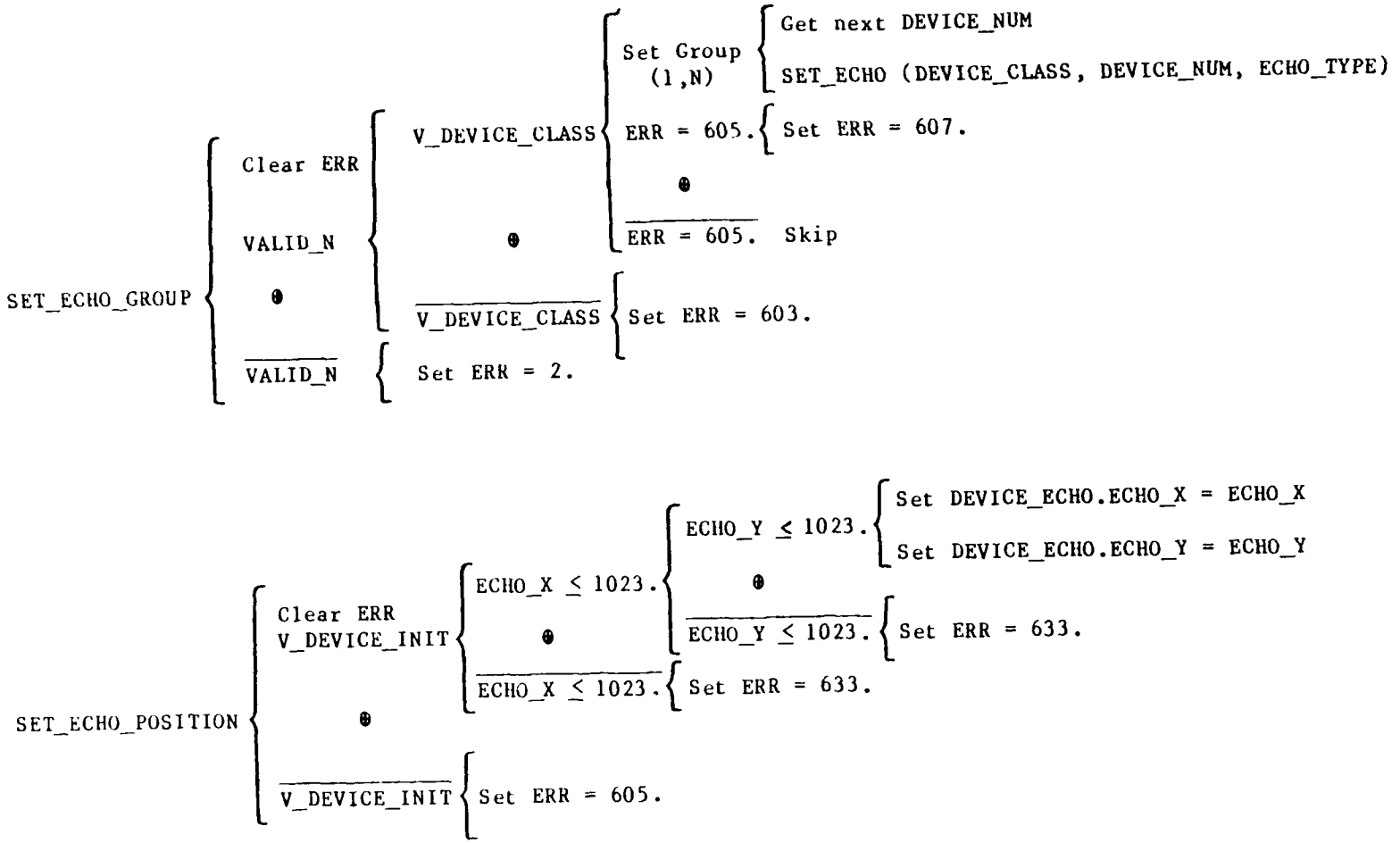
```
V_ECHO { ECHO_TYPE Valid for DEVICE_CLASS { Return TRUE
         •
         ECHO_TYPE Valid for DEVICE_CLASS { Return FALSE
```



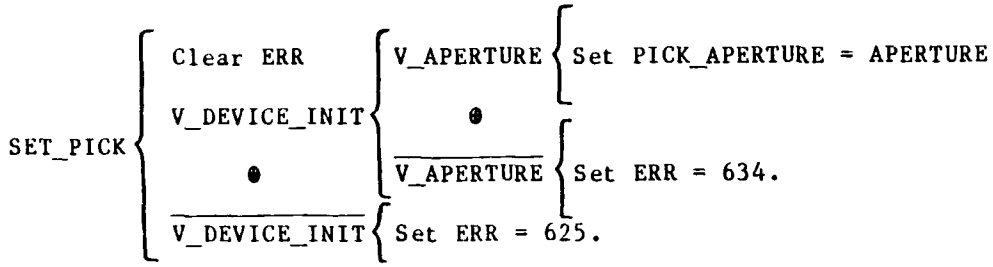

GET_STRING (INPUT_STRING, LENGTH, TIME_OUT)


START_KEYBOARD

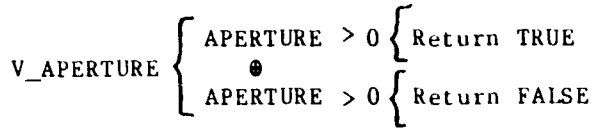

AWAIT_ANY_BUTTON_GET_LOCATOR-2
Clear ERR
V_DEVICE_INIT (LOCATOR)
V_DEVICE_INIT (LOCATOR)
AWAIT_ANY_BUTTON (TIME, BUTTON_NUM)
ERR > 0
Skip
ERR > 0
READ_LOCATOR_2 (LOCATOR_NUM, X, Y)
Return
Set ERR = 628.
AWAIT_ANY_BUTTON_GET_VALUATOR
Clear ERR
V_DEVICE_INIT (VALUATOR)
V_DEVICE_INIT (VALUATOR)
AWAIT_ANY_BUTTON (TIME, BUTTON_NUM)
ERR > 0
Skip
ERR > 0
READ_VALUATOR (VALUATOR_NUM, VALUE)
Return
Set ERR = 629.
SET_ECHO
V_DEVICE_INIT
V_DEVICE_INIT
V_ECHO
Set DEVICE_ECHO = ECHO_TYPE
V_ECHO
Set ERR = 630.
Set ERR = 605.

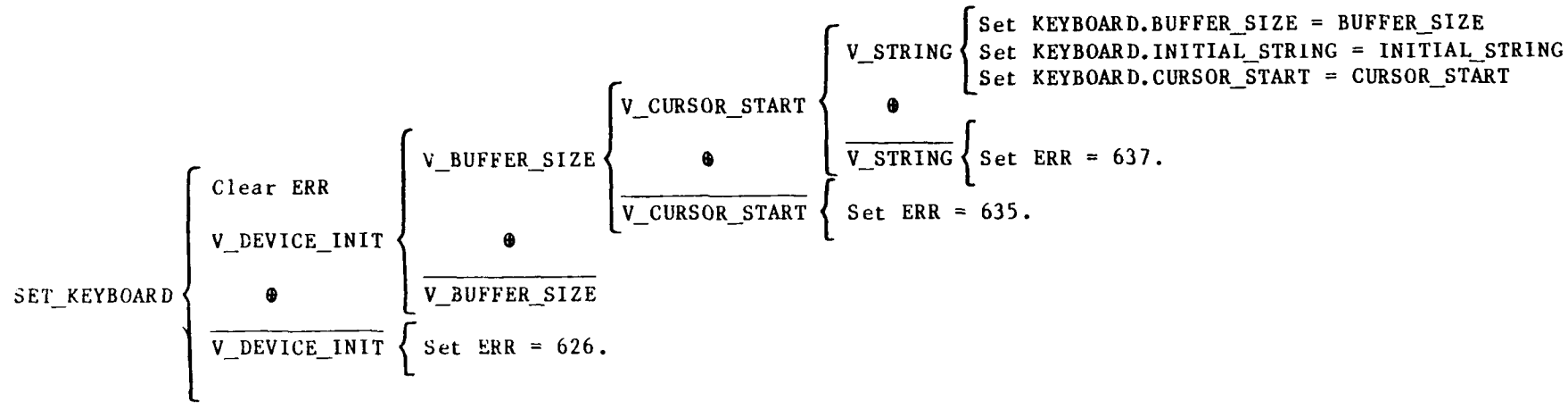

SET_ECHO_GROUP
Clear ERR
VALID_N
VALID_N
V_DEVICE_CLASS
V_DEVICE_CLASS
Set Group (1,N)
Get next DEVICE_NUM
SET_ECHO (DEVICE_CLASS, DEVICE_NUM, ECHO_TYPE)
ERR = 605.
Set ERR = 607.
ERR = 605. Skip
Set ERR = 603.
Set ERR = 2.


SET_ECHO_POSITION
Clear ERR
V_DEVICE_INIT
V_DEVICE_INIT
ECHO_X ≤ 1023.
ECHO_X ≤ 1023.
ECHO_Y ≤ 1023.
ECHO_Y ≤ 1023.
Set DEVICE_ECHO.ECHO_X = ECHO_X
Set DEVICE_ECHO.ECHO_Y = ECHO_Y
Set ERR = 633.
Set ERR = 633.
Set ERR = 605.

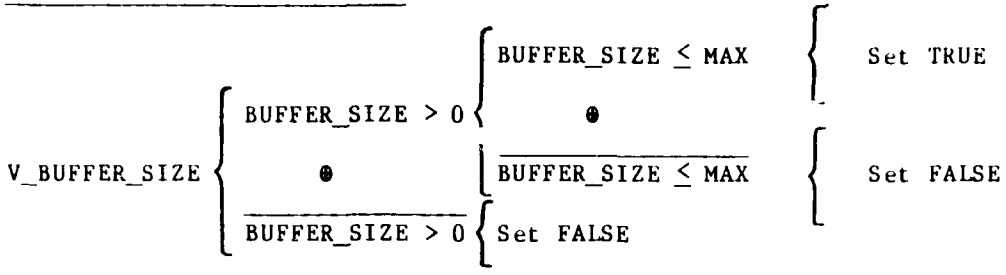



V_APERTURE (APERTURE)

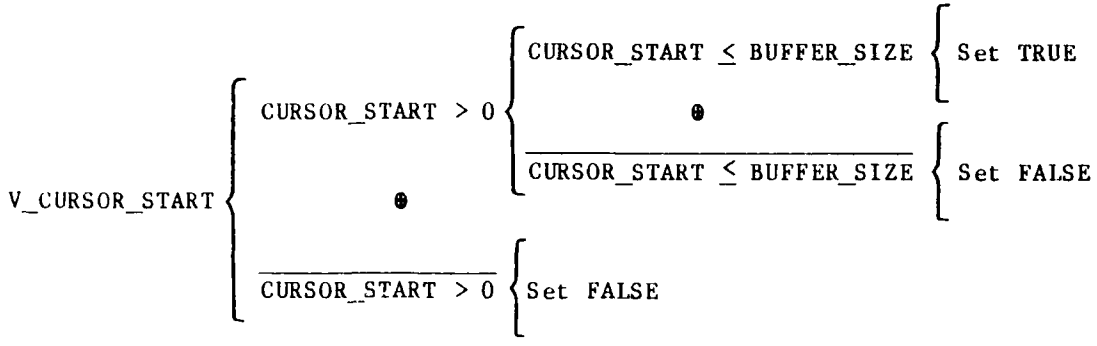


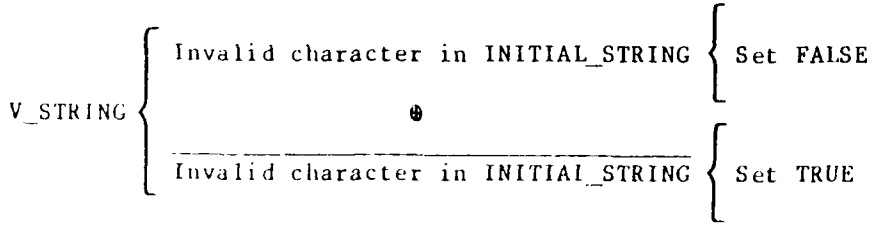
V_BUFFER_SIZE (BUFFER_SIZE)




V_CURSOR_START (CURSOR_START, BUFFER_SIZE)



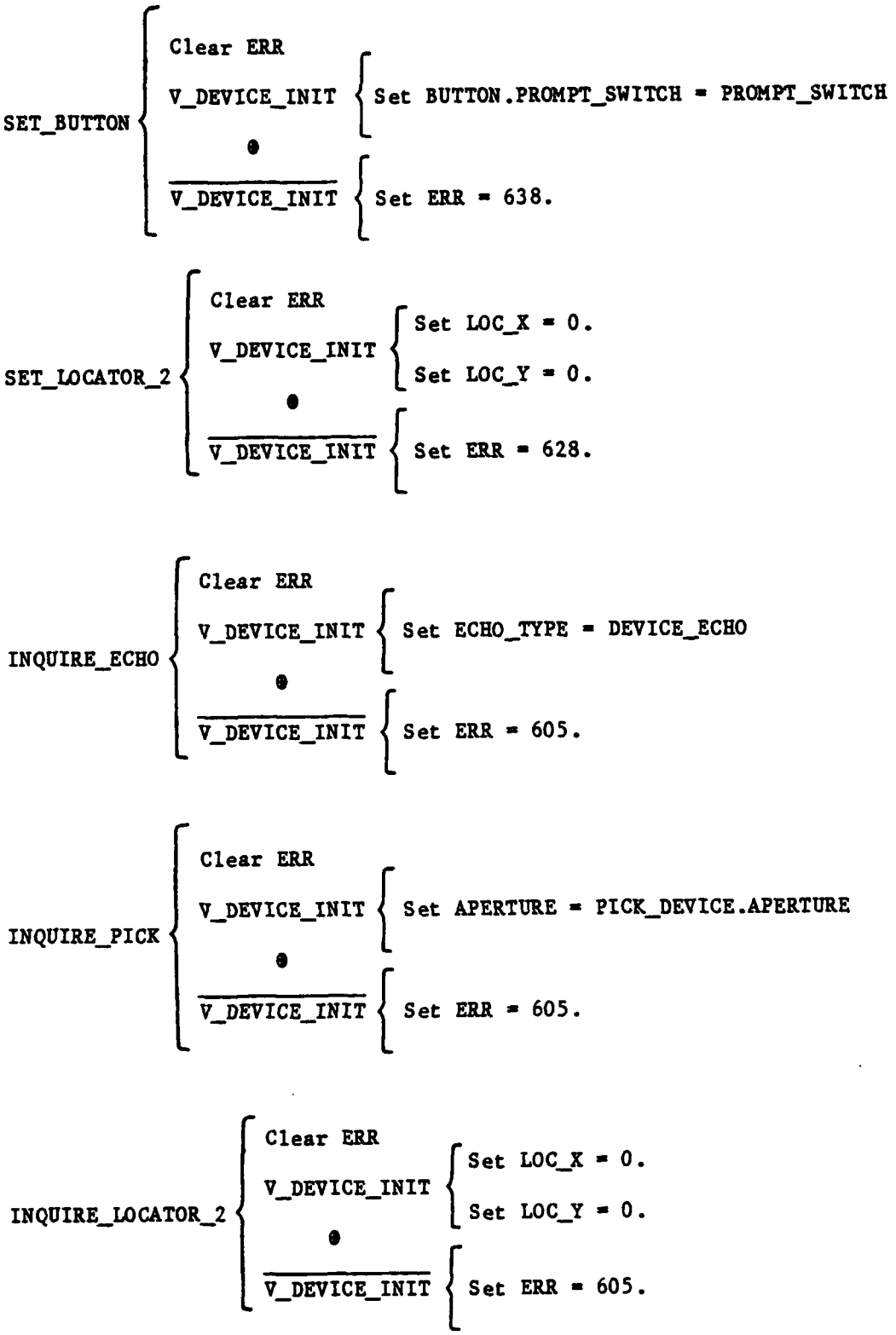

V_STRING (INITIAL_STRING)

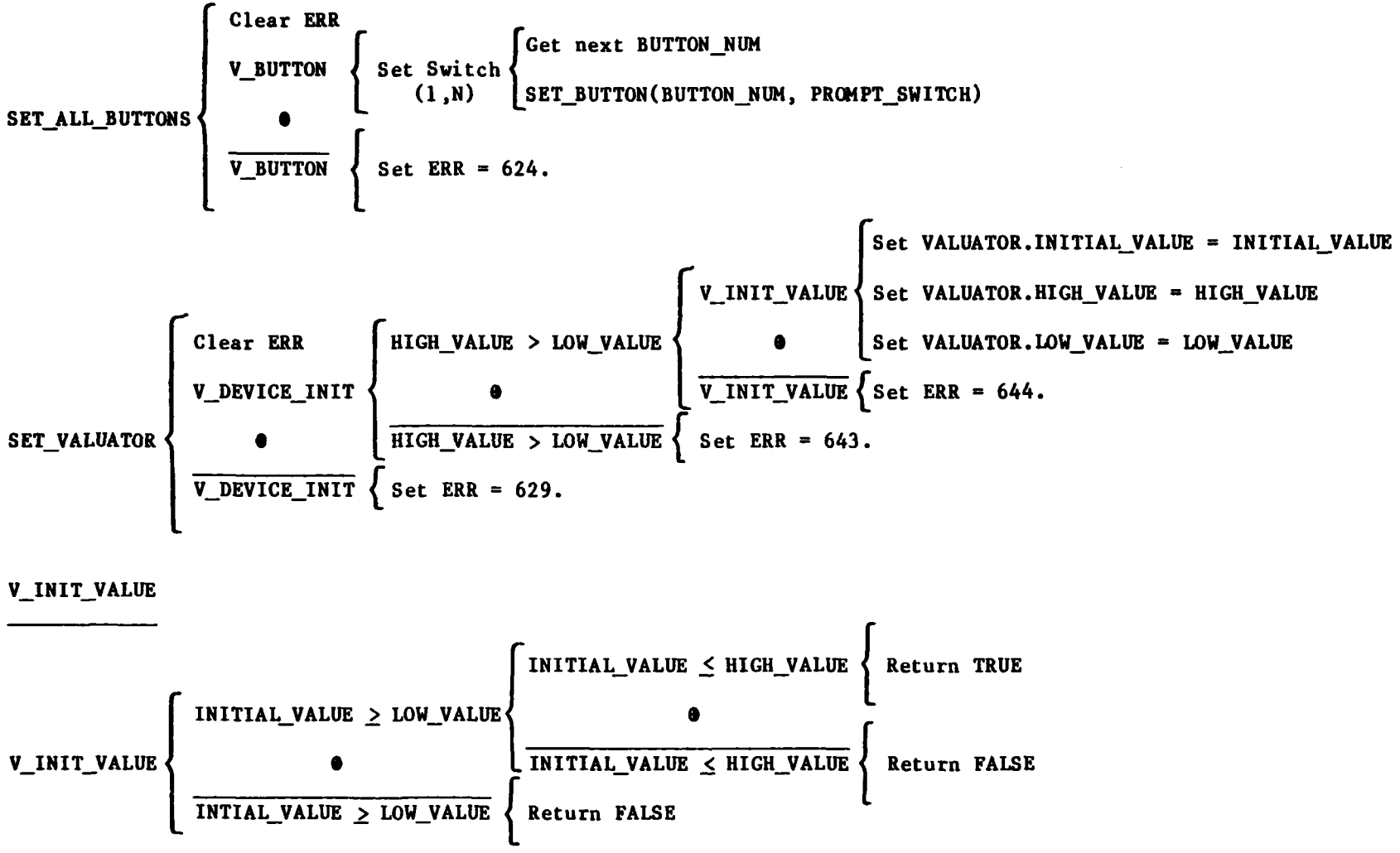


AD-A100 880 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM--ETC(U)
DEC 80 H L CURLING
UNCLASSIFIED AFIT/GCS/EE/80D-6 NL

2 OF 2
AD A
100880

END
DATE FILMED
7-81
DTIC


SET_BUTTON
Clear ERR
V_DEVICE_INIT
Set BUTTON.PROMPT_SWITCH = PROMPT_SWITCH
•
V_DEVICE_INIT
Set ERR = 638.
SET_LOCATOR_2
Clear ERR
V_DEVICE_INIT
Set LOC_X = 0.
Set LOC_Y = 0.
•
V_DEVICE_INIT
Set ERR = 628.
INQUIRE_ECHO
Clear ERR
V_DEVICE_INIT
Set ECHO_TYPE = DEVICE_ECHO
•
V_DEVICE_INIT
Set ERR = 605.
INQUIRE_PICK
Clear ERR
V_DEVICE_INIT
Set APERTURE = PICK_DEVICE.APERTURE
•
V_DEVICE_INIT
Set ERR = 605.
INQUIRE_LOCATOR_2
Clear ERR
V_DEVICE_INIT
Set LOC_X = 0.
Set LOC_Y = 0.
•
V_DEVICE_INIT
Set ERR = 605.

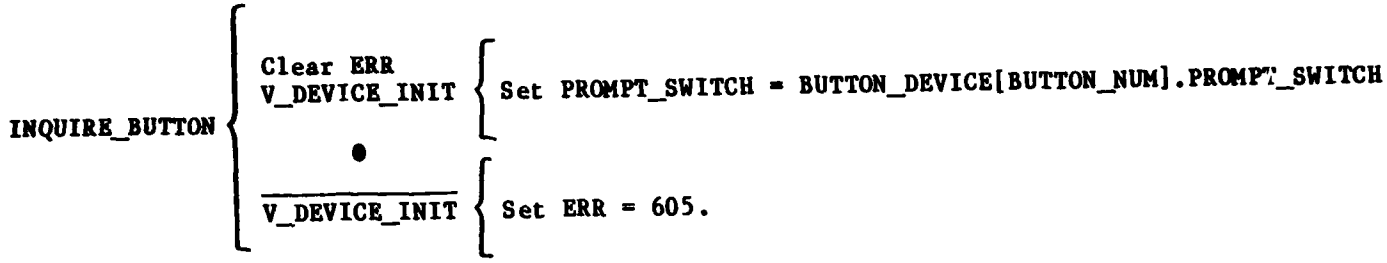

SET_ALL_BUTTONS
Clear ERR
V_BUTTON
Set Switch (1,N)
Get next BUTTON_NUM
SET_BUTTON(BUTTON_NUM, PROMPT_SWITCH)
●
V_BUTTON
Set ERR = 624.
SET_VALUATOR
Clear ERR
V_DEVICE_INIT
●
V_DEVICE_INIT
Set ERR = 629.
HIGH_VALUE > LOW_VALUE
●
HIGH_VALUE > LOW_VALUE
Set ERR = 643.
V_INIT_VALUE
●
V_INIT_VALUE
Set ERR = 644.
Set VALUATOR.INITIAL_VALUE = INITIAL_VALUE
Set VALUATOR.HIGH_VALUE = HIGH_VALUE
Set VALUATOR.LOW_VALUE = LOW_VALUE
V_INIT_VALUE
V_INIT_VALUE
INITIAL_VALUE ≥ LOW_VALUE
●
INTIAL_VALUE ≥ LOW_VALUE
Return FALSE
INITIAL_VALUE ≤ HIGH_VALUE
Return TRUE
●
INITIAL_VALUE ≤ HIGH_VALUE
Return FALSE

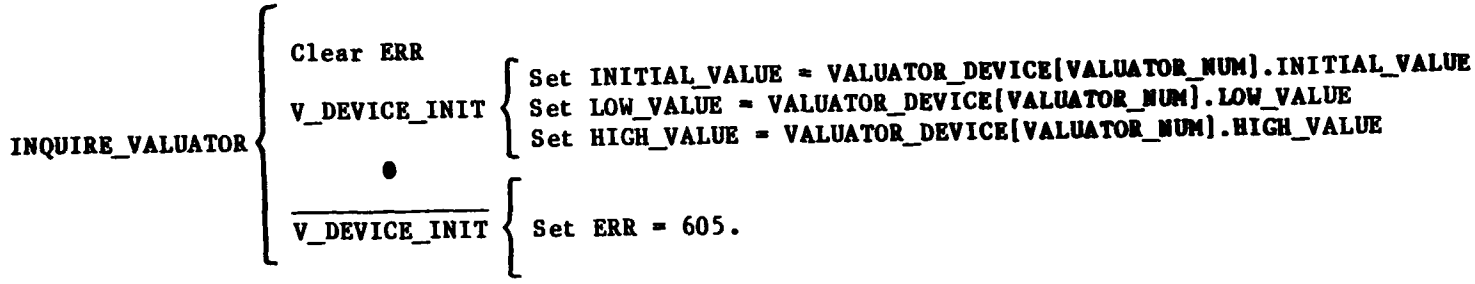
```
INQUIRE_INPUT_CAPABILITIES
    Set LEVEL = 2
    Set DEVICE_COUNTS
        Set PICK_COUNT = 1
        Set KEYBOARD_COUNT = 1
        Set BUTTON_COUNT = 8
        Set STROKE_COUNT = 0
        Set LOCATOR_COUNT = 1
        Set VALUATOR_COUNT = 4
    Set TIMING = 2

INQUIRE_LOCATOR_DIMENSION
    Clear ERR
    V_DEVICE_INIT
        Set DIMENSION = '2D'
    •
    ~V_DEVICE_INIT~
        Set ERR = 605.

INQUIRE_DEVICE_STATUS
    Clear ERR
    V_DEVICE_CLASS
        V_DEVICE_NUM
            Set INITIALIZED = DEVICE_INIT
            Set ENABLED = DEVICE_ENABLE
        •
        ~V_DEVICE_NUM~
            Set ERR = 602.
    •
    ~V_DEVICE_CLASS~
        Set ERR = 602.
```

```
INQUIRE_ECHO_POSITION
    Clear ERR
    V_DEVICE_INIT
        Set ECHO_X = DEVICE_CLASS[DEVICE_NUM].ECHO_POSITION.ECHO_X
        Set ECHO_Y = DEVICE_CLASS[DEVICE_NUM].ECHO_POSITION.ECHO_Y
    •
    ~V_DEVICE_INIT
        Set ERR = 605.
```

```
INQUIRE_ECHO_SURFACE
    Clear ERR
    V_DEVICE_INIT
        ~DEVICE_CLASS = 'STROKE'
            Set SURFACE_NAME = 0
        •
        DEVICE_CLASS = 'STROKE'
            Set ERR = 801.
    •
    ~V_DEVICE_INIT
        Set ERR = 605.
```

```
INQUIRE_KEYBOARD
    Clear ERR
    V_DEVICE_INIT
        Set BUFFER_SIZE = KEYBOARD_DEVICE.BUFFER_SIZE
        Set INITIAL_STRING = KEYBOARD_DEVICE.INITIAL_STRING
        Set CURSOR_START = KEYBOARD_DEVICE.CURSOR_START
    •
    ~V_DEVICE_INIT
        Set ERR = 605.
```


INQUIRE_BUTTON
Clear ERR
V_DEVICE_INIT
Set PROMPT_SWITCH = BUTTON_DEVICE[BUTTON_NUM].PROMPT_SWITCH
•
V_DEVICE_INIT
Set ERR = 605.


INQUIRE_VALUATOR
Clear ERR
V_DEVICE_INIT
Set INITIAL_VALUE = VALUATOR_DEVICE[VALUATOR_NUM].INITIAL_VALUE
Set LOW_VALUE = VALUATOR_DEVICE[VALUATOR_NUM].LOW_VALUE
Set HIGH_VALUE = VALUATOR_DEVICE[VALUATOR_NUM].HIGH_VALUE
•
V_DEVICE_INIT
Set ERR = 605.

## INPUT SYSTEM FUNCTIONS NOT SUPPORTED

The primitives on the following pages of this appendix are not supported by this design because of three basic reasons :

1. The input system designed in this effort is synchronous only. It will not support any asynchronous primintives (e.g., AWAIT_EVENT, device associations or device event report data acquisition). This is a restriction due to the terminal (4014) used.

2. The system designed does not support '3D' primitives. This restriction is also due to the terminal used.

3. The system designed is not yet integrated with an OUTPUT CORE system. Therefore, any reference to SURFACES, PORTS, SEGMENTS, etc. are not supported. These primitives will be supportable when the OUTPUT CORE system is designed and is ready to be integrated with this design.

```
READ_LOCATOR_3 { Return ERR = 801.
AWAIT_EVENT { Return ERR = 801.
FLUSH_DEVICE_EVENTS { Return ERR = 801.
FLUSH_GROUP_EVENTS { Return ERR = 801.
FLUSH_ALL_EVENTS { Return ERR = 801.
ASSOCIATE { Return ERR = 801.
DISASSOCIATE { Return ERR = 801.
DISASSOCIATE_DEVICE { Return ERR = 801.
DISASSOCIATE_GROUP { Return ERR = 801.
DISASSOCIATE_ALL { Return ERR = 801.
GET_PICK_DATA { Return ERR = 801.
GET_KEYBOARD_DATA { Return ERR = 801.
GET_STROKE_DATA_2 { Return ERR = 801.
GET_STROKE_DATA_3 { Return ERR = 801.
GET_LOCATOR_DATA_2 { Return ERR = 801.
GET_LOCATOR_DATA_3 { Return ERR = 801.
GET_VALUATOR_DATA { Return ERR = 801.
AWAIT_STROKE_2 { Return ERR = 801.
AWAIT_STROKE_3 { Return ERR = 801.
AWAIT_ANY_BUTTON_GET_LOCATOR_3 { Return ERR = 801.
SET_ECHO_SEGMENT { Return ERR = 801.
SET_ECHO_SURFACE { Return ERR = 801.
SET_STROKE { Return ERR = 801.
SET_LOCATOR_3 { Return ERR = 801.
SET_LOCPORT_2 { Return ERR = 801.
SET_LOCPORT_3 { Return ERR = 801.
INQUIRE_DEVICE_ASSOCIATIONS { Return ERR = 801.
```

```
INQUIRE_STROKE { Return ERR = 801.
INQUIRE_LOCATOR_3 { Return ERR = 801.
INQUIRE_LOCPORT_2 { Return ERR = 801.
INQUIRE_LOCPORT_3 { Return ERR = 801.
INQUIRE_ECHO_SEGMENTS { Return ERR = 801.
```

## Appendix D

The purpose of this appendix is to supply some miscillaneous information about the VAX/VMS PASCAL compiler. The compiler supports all standard PASCAL features (as in Jensen and Wirth). It also supports some no-standard features which are beneficial to this investigation and to the follow-on work of implementing this design.

The two most important of these features and the ones which influenced this design are:

1. EXTERNAL Subprograms

2. TIME

EXTERNAL Subprograms - EXTERNAL subprograms are those subroutines which are compiled or assembled separately from the routine from which they are invoked. VAX PASCAL handles these routines in either of two ways : by declaration as a FORTRAN procedure/function or by declaration as an EXTERNAL procedure/function. However, these two declarations are identical. Section 6.7 of the VAX-11 PASCAL Language Reference Manual [Ref 5] gives a detailed description of how to use these declarations.

TIME - TIME is a utility function which returns a character string with the current time to one hundreth of a second. The argument passed to the function must be declared as a PACKED

ARRAY[1..11] OF CHARacter.

One other point needs to be discussed : The SYS$QIO calls for interrupt-driven routines (e.g., START_PICK, START_BUTTON, etc.). A complete discussion of SYS$QIO can be found in the VAX/VMS System Services Reference Manual [Ref 4 :138-143 and Chapter 6]. Briefly, the SYS$QIO utility allows the user to 'queue' an I/O request to a device through the system and then return to the program and continue processing. The completion of the request causes one of three results. The user must poll the system for device status to determine if the request has been serviced, direct the system to interrupt his/her program at request completion, or wait for a specified event flag to be set. It is the second of these alternatives which is used in this design to accomplish TIMEd operations in the event handling routines.

An important point to remember is that a call to the system utility SYS$ASSIGN must be made in order to obtain the device CHANnel number. Care must also be taken when specifying the formal parameters in the utility calls : some parameters require immediate translation (no evaluation) while others require an evaluation (VALUE) in order to be passed to the utility routine correctly. The VAX/VMS manual details these points.

## Vita

Harold Leon Curling, Jr. was born on 25 October 1948 in Norfolk, Virginia. He graduated from Princess Anne High School in Virginia Beach, Virginia in 1966. He attended the Virginia Polytechnic Institute and State University and Colorado State University from which he received the Bachelor of Science degree in Electrical Engineering in May 1975. He enlisted in the Air Force in April of 1969 and was assigned to the Air Force Technical Applications Center at Ascension Island and Sunnyvale, California from May 1970 to May 1973. He was then accepted into the Air Force Institute of Technology's Airman's Education and Commissioning Program and assigned to Colorado State University from June 1973 to July 1975. He then attended the Air Force Officer Training School at Lackland AFB, Texas from July to October 1975 and was awarded an Air Force commission upon graduation. From there he was assigned to the Air Force Weapons Laboratory's Airborne Laser Laboratory as the laser device controls engineer. He entered the Air Force Institute of Technology at Wright-Patterson AFB, Ohio in June 1979.

Permanent address: 3320 Justis Street

Virginia Beach, Virginia 23462

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>AFIT/GCS/EE/80 D-6 | 2. GOVT ACCESSION NO.<br>AD-A100 880 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>DESIGN OF AN INTERACTIVE INPUT GRAPHICS SYSTEM BASED ON THE ACM CORE STANDARD | | 5. TYPE OF REPORT & PERIOD COVERED<br>Master of Science Thesis<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>HAROLD L. CURLING, JR., Captain, USAF | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Air Force Institute of Technology (AFIT/EN)<br>Wright-Patterson AFB, Ohio 45433 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS | | 12. REPORT DATE<br>December 1980<br>13. NUMBER OF PAGES<br>99 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified<br>15a. DECLASSIFICATION DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

FREDERIC C. LYNCH, Major, USAF
Director of Public Affairs

11 JUN 1981

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Graphics
CORE Standard
Interactive Graphics
Warnier-Orr Diagrams

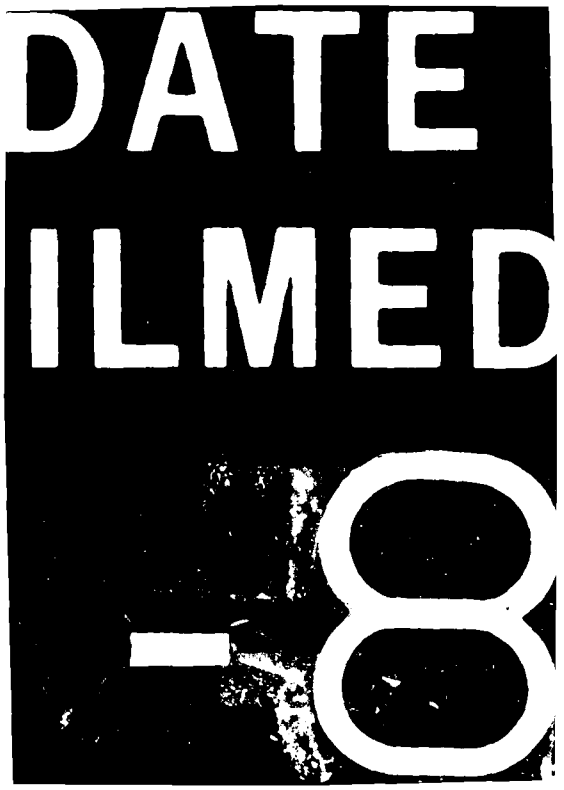
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A design has been developed for the interactive input section of a graphics system based on the ACM/SIGGRAPH CORE standard of 1979. The input system has been designed for use on a Digital Equipment Corporation VAX 11/780 computer within the VAX/VMS operating system and contains the design for a device driver for the Tektronix 4014 graphics terminal. The design is readily expandable to include software drivers for other graphics devices.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The Warnier-Orr diagram method of software design has been used as the vehicle for the development of the design. Three dimensional functions, the STROKE logical device and asynchronous event generation and data access are not supported by the design. However, all input functions required by the CORE standard have been included in the design.


DATE
ILMED
-8